



# **Das 68000-Paket**

Benutzer-Handbuch  
OPAL-68000



```
*****
*                               *
*   Opal-68000  Cross-Assembler   *
*                               *
*           Vers. 1.03             *
*                               *
*   Bedienungsanleitung           *
*                               *
*****
```

(C) - 1984 Wilke / IDA-Software

Stand: 17.8.84 (1)

## C o p y r i g h t

=====

"Des 68000-Paket", bestehend aus Computer-Programmen und schriftlichen Unterlagen ist geistiges Eigentum des Lizenz-Inhabers, Hans-Jürgen Wilke, 5100 Aachen, Postfach 1727. Diesen Sachverhalt erkennen Händler und End-Abnehmer dieses Produktes an.

Mit Zahlung des 'Kaufpreises' entrichtet der End-Abnehmer eine Lizenzgebühr, die ihn dazu berechtigt, diese Programme auf einem Computer ablaufen zu lassen und entsprechend dem hier beschriebenen Verwendungszweck zu benutzen (Einzel-Benutzerrecht). Dieses Recht ist nicht übertragbar, weitere Rechte bedürfen der schriftlichen Vereinbarung mit dem Lizenz-Inhaber.

Nicht gestattet sind insbesondere:

- das Kopieren des Produktes, oder Teilen hiervon, außer zum Zwecke der persönlichen Programmsicherung (Backup),
- die Weitergabe des Produktes oder Kopien hiervon, im Ganzen oder in Teilen,
- die Veränderung des Produktes oder Überführung in eine andere Darstellungsform, also z.B.:
  - des Listens, Disassemblierens, Decompilieren, Übersetzen in andere Sprachen, die Überführung in ein anderes elektronisches oder nichtelektronisches Aufzeichnungs-Verfahren.
- Das gleichzeitige Benutzen dieses Produktes auf mehreren Computer-Anlagen.

Der Händler erwirbt kein Benutzerrecht an diesem Produkt, vielmehr tritt er gegenüber dem Endbenutzer als Vermittler auf, der für seine Tätigkeit eine Vermittler-Provision (Handels-spenne) erhält.

## B e w e h r l e i s t u n g s a u s s c h l u ß

=====

Der Lizenz-Inhaber behält sich vor, Änderungen an diesem Produkt vorzunehmen ohne die Verpflichtung diese irgendjemandem bekanntzugeben. Ferner ist jede Schadenersatz-Forderung an den Lizenz-Inhaber ausgeschlossen, falls im Zusammenhang mit diesem Produkt Kosten oder sonstige Schäden entstehen.

## Übersicht =====

Die Bedienungs-Anleitung ist in 3 Teile aufgeteilt:

- |                                  |           |
|----------------------------------|-----------|
| 1.) Beschreibung kurz & bündig:  | Seite 6   |
| 2.) Beschreibung mit Beispielen: | Seite 12  |
| 3.) Musterprogramm-Listings:     | Seite 127 |

Kapitel 1 ist eine sehr knappe und übersichtliche Darstellung, die ist vor allen Dingen für das schnelle Auffinden bestimmter Details gedacht. Nach einer gewissen Einarbeitungszeit dürfte dieser Teil des Handbuchs der meist benutzte sein.

Zum Kennenlernen ist das Kapitel 2 gedacht. Nach einigen Vorbemerkungen wird in 2.2. sofort ein kleines Muster-Programm assembliert, an Hand dessen die Bedienung des Assemblers schnell deutlich wird.

Schließlich sind in Kap. 3 einige Muster-Programme enthalten, in denen die Wirkung verschiedener Anweisungen demonstriert wird.

## Inhaltsverzeichnis

### 1. Kurzbeschreibung

1.1. Starten des Assemblers	... S.	5
1.2. Command-Line Switches	... S.	6
1.3. Namens-Vereinbarungen	... S.	6
1.4. Assembler-Anweisungen (Pseudo Opcodes)	... S.	6
1.5. MC-68000 Opcodes	... S.	7
1.6. Adressierungsarten, Syntax	... S.	9
1.7. Operatoren, Ausdrücke	... S.	10
1.8. Listing-Format	... S.	11
1.9. Fehler-Meldungen	... S.	11

### 2. Beschreibung

2.1. Allgemeine Eigenschaften, Voraussetzungen	... S.	12
2.2. Assemblieren eines Muster-Programms	... S.	14
2.3. Command-Line Switches, Filenamen	... S.	16
2.4. Assembler-Anweisungen	... S.	18
2.4.1. XLIST	... S.	18
2.4.2. LIST	... S.	18
2.4.3. PAGE	... S.	19
2.4.4. TOP	... S.	19
2.4.5. LINE	... S.	20
2.4.6. LINIT	... S.	20
2.4.7. LEXIT	... S.	21
2.4.8. XPUNCH	... S.	22
2.4.9. PUNCH	... S.	22
2.4.10. TITLE	... S.	23
2.4.11. XFLAG	... S.	24
2.4.12. FLAG	... S.	24
2.4.13. DC	... S.	25
2.4.14. DS	... S.	27
2.4.15. FILL	... S.	27
2.4.16. EVEN	... S.	28
2.4.17. EOU	... S.	29
2.4.18. ORG	... S.	30
2.4.19. SIZE	... S.	30
2.4.20. PRINT	... S.	31
2.4.21. INPUT	... S.	32
2.4.22. IFE, IFN, IFP, IFM	... S.	33
2.4.23. ENDF	... S.	37
2.4.24. INCLUDE	... S.	37
2.4.25. REDEF	... S.	37

2.5. Adressierungsarten, Syntax	... S. 38
2.6. Symbole, Konstanten, Operatoren, Ausdrücke	... S. 40
2.7. Fehler-Meldungen	... S. 42
2.8. MC-68000 Opcodes	... S. 43
 3. Muster-Programme	 ... S. 127

## 1. Kurzbeschreibung

### 1.1. Starten des Assemblers

A) OPAL FILENAME(.EXT) [/Switch-1 [/Switch-2 .. ]]

Während des Assemblerlaufs ist Abbruch durch CTRL-C möglich.

### 1.2. Command-Line Switches

			Defaults:
/Px	1 x = A-M -->	Drive für Listing-File	'source-dr'
	1 x = N -->	kein Listing	
	1 x = P -->	Listing an Printer	
	1 x = X -->	Listing an Console	
	1 x = ,Y -->	Listing an AUX-Output	
/F	:	nur fehlerhefte Zeilen listen	
/Dx	1 x = A-M -->	Drive für Object-File	'source-dr'
	1 x = N -->	kein Object-File erzeugen	
/Ex	1 x = A-M -->	Drive für EPROM-Daten	'source-dr'
	1 x = N -->	kein EPROM-Daten File	

### 1.3. Namens-Vereinbarungen

Namen im Rahmen der Betriebs-System-Vorgaben frei wählbar, für die Namens-Erweiterung gilt:

.H68	=	68000-Source Code	(Default)
.LST	=	Listing-File	(inner)
.COD	=	Object-File	(inner)
.BIN	=	Binär-File f. EPROM	(inner)

### 1.4. Pseudo-Opcodes

			Defaults:
LIST	- ohne Arg.:	schaltet Listing ein	ein
XLIST	- ohne Arg.:	schaltet Listing aus	-
PAGE	- ohne Arg.:	neue Seite	-
	mit Arg.:	setzt Seitenlänge	68
TOP	- mit Arg.:	setzt Top-Lines	4
LINE	- mit Arg.:	setzt Zeilenlänge	79
INIT	- mit Arg.:	Drucker-Init-Sequenz	-
EXIT	- mit Arg.:	Drucker-Exit-Sequenz	-
PUNCH	- ohne Arg.:	drucke Maschinen-Code aus	ein
XPUNCH	- ohne Arg.:	unterdrücke Maschinen-Code	-
TITLE	- mit Arg.:	Titel-Zeile für Listing	-
FLAG	- ohne Arg.:	zeige Flags im Listing an	ein
XFLAG	- ohne Arg.:	unterdrücke Listing-Flags	-
DC.X	- mit Arg.:	Definiere Konstanten	-
		.X = opt. Size-Angabe	
DS.X	- mit Arg.:	Definiere Speicherbereich	-
		.X = opt. Size-Angabe	
FILL	- mit Arg.:	setzt Fill-Zeichen für 'DS.X'	00H
EVEN	- ohne Arg.:	PC auf nächste gerade ADR	-



## Fortsetzung:

## 1.4. Pseudo-Opcodes

			Defaults:
EQU	- mit Arg.:	Symbol-Definition	-
DAB	- mit Arg.:	Adress-Definition	-
SIZE.X	- ohne Arg.:	setzt Default-Size	LONG
PRINT	- mit Arg.:	Drucke Argument (Pass-1)	-
INPUT	- ohne Arg.:	hole Wert von Benutzer (Pass-1)	-
IFE	- mit Arg.:	Beginn conditional-Assembly	-
IFN	- mit Arg.:	Beginn conditional-Assembly	-
IFP	- mit Arg.:	Beginn conditional-Assembly	-
IFM	- mit Arg.:	Beginn conditional-Assembly	-
ENDIF	- ohne Arg.:	beendet conditional-Assembly	-
INCLUDE	- mit Arg.:	liest Text-File ein	-
REDEF	- mit Arg.:	Redefinition eines Symbols	-

## 1.5. MC-68000 Dpcodes

ABCB.X	op1,op2	- Add Decimel with Extend
ADD.X	op1,op2	- Add Binery
ADDA.X	op1,op2	- Add Address
ADDI.X	op1,op2	- Add immediate
ADDQ.X	op1,op2	- Add Quick
ADDX.X	op1,op2	- Add Extended
AND.X	op1,op2	- Logical AND
ANDI.X	op1,op2	- Logical AND Immediate
ASL.X	op1(,op2)	- Arithmetic Shift Left
ASR.X	op1(,op2)	- Arithmetic Shift Right
Bcc.X	op1	- Branch Conditionally
BCHG.X	op1,op2	- Test a Bit and Change
BCLR.X	op1,op2	- Test a Bit and Clear
BRA.X	op1	- Branch
BSET.X	op1,op2	- Test a Bit and Set
BSR.X	op1	- Branch to Subroutine
BTST.X	op1,op2	- Test a Bit
CHK.W	op1,op2	- Check Register Against Bounds
CLR.X	op1	- Clear an Operand
CMP.X	op1,op2	- Compare
CMFA.X	op1,op2	- Compare Address
CMP1.X	op1,op2	- Compare Immediate
CMPI.X	op1,op2	- Compare Memory
DBcc.W	op1,op2	- Test, Decrement and Branch
DIVS.W	op1,op2	- Divide with Sign
DIVU.W	op1,op2	- Divide Unsigned
EOR.X	op1,op2	- Logical Exclusive OR
EDR1.X	op1,op2	- Logical Exclusive OR Immediate
EXG.L	op1,op2	- Exchange Registers
EXT.X	op1	- Sign Extend

Fortsetzung:

## 1.5. MC-68000 Opcodes

JMP	op1	-	Jump
JSR	op1	-	Jump to Subroutine
LEA.L	op1,op2	-	Load Effective Address
LINK	op1,op2	-	Link and Allocate Stack
LSL.X	op1,op2	-	Logical Shift Left
LSR.X	op1,op2	-	Logical Shift Right
MOVE.X	op1,op2	-	Move Data
MOVEA.X	op1,op2	-	Move Address
MOVEM.X	op1,op2	-	Move Multiple Registers
MOVEP.X	op1,op2	-	Move Peripheral Data
MOVEQ.L	op1,op2	-	Move Quick
MULS.W	op1,op2	-	Multiply with Sign
MULU.W	op1,op2	-	Multiply Unsigned
NBCD.B	op1,op2	-	Negate Decimal with Extend
NEG.X	op1	-	Negate
NEGX.X	op1	-	Negate with Extend
NOP		-	No Operation
NOT.X	op1	-	Logical Not
OR.X	op1,op2	-	Logical Or
ORI.X	op1,op2	-	Logical Or Immediate
PEA.L	op1	-	Push Effective Address
RESET		-	Reset External Devices (privil.)
ROL.X	op1(,op2)	-	Rotate Left
ROR.X	op1(,op2)	-	Rotate Right
ROXL.X	op1(,op1)	-	Rotate Left with Extend
ROXR.X	op1(,op2)	-	Rotate Right with Extend
RTE		-	Return from Exception (privil.)
RTR		-	Return and Restore CCR
RTS		-	Return from Subroutine
SBCD.B	op1,op2	-	Subtract Decimal with Extend
SEC.B	op1	-	Set According to Condition
STDP	op1	-	Load SR and Stop
SUB.X	op1,op2	-	Subtract Binary
SUBA.X	op1,op2	-	Subtract Address
SUBI.X	op1,op2	-	Subtract Immediate
SUBQ.X	op1,op2	-	Subtract Quick
SUBX.X	op1,op2	-	Subtract with Extend
SWAP.W	op1	-	Swap Register Helves

Fortsetzung:

### 1.5. MC-68000 Opcodes

TAS.B	opl	- Test and Set on Operand
TRAP	opl	- Trap
TRAPV		- Trap on Overflow
TST.X	opl	- Test on Operand
UNLK	opl	- Unlink

Die Opcodes: ADDA, CMPA, CMPI, MOVEA, SUBA und SUBI

können auch durch: ADD, CMP, MOVE und SUB

abgekürzt werden.

### 1.6. Addressing-Modes, Syntax

Dn	:	D3	
An	:	A1	
(An)	:	(A2)	
(An)+	:	(A3)+	
-(An)	:	-(A4)	
d(An)	:	OFFSET (A3)	: -12X8 +SYMBOL(A2)
d(An,Ri)	:	LABEL (A1,D2.W)	: NEAR_BY +3 (A2,D2.L)
Abs.W	:	@ LABEL.W	
Abs.L	:	@ LABEL.L	: @LABEL
d(PC)	:	DISPLACEMENT (#)	
d(PC,Ri)	:	SMALL_DISP (#, D6.L)	
Imm	:	@ (VIEL-3+K7) * SYMBOL	

CCR	-	Condition-Code Register (Flags)
SR	-	Status-Register
USP	-	User Stackpointer

Reg-List - A2/A1/D1/D6/D3

Symbol-Definition:	'i'	SYMBOL1
Symbol-Zeichen:	'A..Z', '0..9', '_'	NAME_1
Kommentar-Definition:	'i'	I don't care ...
String-Definition:	'''	'String'
Zahlenbasis 2:	'X2'	1010001111X2
Zahlenbasis 8:	'XB'	77212601:8
Zahlenbasis 10:	-	3124
Zahlenbasis 16:	'H'	0A4FF5BDH
oder:		0A4FF5BD

## 1.7. Operatoren, Ausdrücke

* / ?	=	Multi, Divi, Rest
- +	=	Plus, Minus
! & %	=	XOR, AND, OR

(nach absteigender Priorität geordnet)

Klammerung zur Veränderung der Prioritäten mit eckigen Klammern:

[ [ KLEIN + GRDSS ] \* 4 - WERT\_2 ] &amp; MASKE

Bei der Verwendung von String-Ausdrücken ist folgende Unterscheidung zu beachten:

'abcdefghi' kann als STRING oder als ZAHL benutzt werden:

'	DC.B	'abcdefghi'	; hier: String
VAR_1	EQU	'abcdefghi'	; hier: Zahlenwert

In diesem beiden Beispielen ist die Verwendung des Stringausdrucks als STRING bzw. als ZAHL eindeutig. Bei anderer Gelegenheit bedarf es einer entsprechenden Festlegung in der Programmzeile:

DC.B	'abcdefghi'	; OK, 9 ASCII-Zeichen
DC.L	'abcdefghi' / 4 + 7	; Fehler !!
DC.L	['abcdefghi'] / 4 + 7	; OK, arithm. Ausdr.

Bei allen Assembler-Anweisungen die Strings zulassen, ist deshalb eine entsprechende Kennzeichnung erforderlich, falls ein String als Zahl verwendet werden soll:

z.B:	0 + 'AB'
	[ 'ABCDE' ]

## Beispiel:

001000	00000EDC	DC.L	[(34 + 'A' + A + 0A] + 'B' - 0 ] / SYMBOL
001004	646365736373	DC.L	'dieses ist ein String' ; String-Interpretation
00100A	206373742063		
001010	696E22537472		
001016	696E67		
001019	017FA902	DC.L	['dies' / 67] ; beachte Klammerung !
0000007E		A7	EQU 126
00000003		SYMBOL:	EQU 3

## 1.8. Listing-Format

Quell-Zeile:

.LABEL:    OPCODE.X    op1,op2    ; Kommentar

Listing-Zeile:

111 222222 333333333333 444 55555555555555555555555555555555

1 = Fehler-Feld, 0-3 Buchstaben kennzeichnen Fehler in Zeile

2 = Adress-Feld, 6 Hex-Ziffern

3 = Code-Feld,    0 - 12 Hex-Ziffern

4 = Flag-Feld,    0 - 3 Flags: N/-/R/C

N-Flag:    Nicht benutztes Symbol

- Flag:    Symbol-REDEF

R-Flag:    Range-Flag, kleinere Size möglich

C-Flag:    INCLUDE ist aktiv

5 = Quelltext-Feld

Aufbau der Symbol-Tabelle:

SYMBOL\_NAME, <Symbol-Wert>, Flags, Art der Definition

SYMBOL\_NAME:       --->   1..12 Zeichen (A..Z, 0..9, \_)

<Symbol-Wert>:      --->   8 Stellen hexadecimal (LONG)

Flags:              --->   M = Fehler, Mehrfach-Definition

                    --->   N = Nicht benutztes Symbol

Art der Definition: --->   equ, redef, label, input

## 1.9. Fehlermeldungen

A = Argument-Fehler / unzul. ADR-Mode  
 I = Illegales Zeichen  
 L = LABEL-Fehler  
 M = Mehrfache Symbol-Definition  
 N = Numerischer Fehler  
 O = Opcode-Fehler  
 P = Phasen-Fehler  
 R = Bereichs-Überschreitung  
 S = SIZE-Fehler  
 U = undefiniertes SYMBOL  
 X = sonst. Fehler

## 2.1. Allgemeine Eigenschaften, Voraussetzungen

DPAL-68000 ist ein Assembler zur Erstellung von 68000 Maschinen-Programmen. Da der Assembler auf Maschinen mit anderen Prozessoren als dem 68000 ebläuft (Z80, 6502, ...), spricht man auch von 'Cross'-Assembler.

Voraussetzung für die sinnvolle Benutzung dieses Assemblers ist ein freier Speicher-Bereich von ca. 32 KBytes (RAM) oder mehr, so daß je nach Installation ca. 500 - 1000 Symbole bearbeitet werden können. Die Länge des zu assemblierenden Programms ist unbegrenzt. Bei vollen 64 KByte RAM können ca. 3000 - 4000 Symbole bearbeitet werden, was üblicherweise einer Quell-Text Länge von mehreren hundert Seiten entspricht.

Die folgende Skizze verdeutlicht die Arbeitsweise des Assemblers:

```
-----
| Quell-Dateien | ==> | Assembler | ==> | Ziel-Dateien |
-----
```

Aus einer oder mehreren Quell-Dateien erzeugt der Assembler 0 - 3 Ziel-Dateien.

Quell-Dateien sind Text-Files mit der Programm-Beschreibung in 68000-Assemblersprache und entsprechend den Konventionen dieses Assemblers. Quell-Dateien sind stets Disk-Files.

Der Assembler-Vorgang wird durch Aufruf des Assemblers mit Angabe eines Quell-Datei Namens gestartet. Mögliche Ziel-Dateien sind:

- Listing
- lauffähiger 68000-Code
- Binär-Code

Je nach Erfordernissen kann der Assembler veranlasst werden, nur die wirklich benötigten Ziel-Dateien zu erzeugen. Die Ziel-Datei 'Listing' kann sowohl als Disk-File als auch als Drucker- bzw. Consolen-Ausgabe erzeugt werden, die beiden anderen Dateien sind stets Disk-Files.

Als Hilfsmittel zur Beeinflussung des Assemblier-Vorganges stehen eine Reihe von Assembler-Anweisungen (Pseudo-OPCODEs) sowie die Commandline-Switches zur Verfügung.

## 2.2. Assemblieren eines Muster-Programms, Aufbau eines Assembler-Quellprogramms.

Zusammen mit dem OPAL-Assembler haben Sie das  
Beispiel-Programm:

HALLD.M6B

bekommen. (---> Kap.3, S 13B)

Um das Programm zu Assemblieren legen Sie  
in Drive A: eine Diskette ein, die sowohl den  
Assembler (DPAL.COM), als auch dieses Beispiel-  
Programm (HALLD.M6B) enthält.

Starten Sie den Assemblerlauf mit:

A>DPAL HALLD.M6B

Nach ein paar Augenblicken sollte sich der Assembler  
melden und schließlich die Meldung:

'Keine Assembler-Fehler'

abgeben.

Ein Blick ins Inhaltsverzeichnis der Diskette zeigt  
die vom Assembler erzeugten Ziel-Dateien:

HALLO.BIN	--> Binär-File
HALLD.CDD	--> lauffähiger Code
HALLD.LST	--> Listing

HALLO.LST ist das Listing und kann direkt auf  
den Bildschirm oder Drucker kopiert werden.  
Die Files .BIN und .CDD enthalten das  
6B000-Maschinen-Programm in zwei verschiedenen  
Formaten.

Löschen Sie diese drei Dateien wieder und versuchen  
Sie mal:

a) A>DPAL HALLO/PN/ON/EN

b) A>DPAL HALLO/PX

c) A>DPAL HALLD/PB

und falls ein Drucker angeschlossen ist:

d) A>DPAL HALLD/PP



zu er: ---> keine Ziel-Datei.  
zu bs: ---> Listing auf Bildschirm.  
zu cs: ---> Listing auf Drive B:  
zu ds: ---> Listing auf Drucker.

Das Listing dieses Muster-Programms ist in Kapitel 3 wiedergegeben und zeigt den grundsätzlichen Aufbau eines 68000-Quellprogramms.

Hervorzuheben ist, daß vor dem ersten 68000-Opcode eine ORG-Anweisung im Programm stehen muß, andernfalls kann keine Assemblierung des Quellprogramms erfolgen.

Der Assembler verarbeitet Quellprogramme zeilenweise. In jeder Zeile kann ein 68000-Opcode oder eine Assembler-Anweisung und ggf. eine LABEL-Definition enthalten sein. LABEL- und SYMBOL-Definitionen werden durch den bündigen Doppelpunkt (:) gekennzeichnet, Kommentare beginnen mit dem Semikolon (;) und enden mit dem Zeilen-Ende.

Feste Spalten-Positionen für die verschiedenen Zeilen-Teile gibt es nicht, je nach Bedarf können Blanke und TABs zur Formatierung verwendet werden.

Beispiel:

```
; Dieses ist ein Kommentar
NOP;Kommentar
LABEL:
NOP
SYMBOL:EQU 12345
LABEL_2:ADDQ.W #3,D5;Kommentar

LONG_LABEL_3:  ADDQ.W  #3,D5           ; Kommentar
                BEQ      overflow
.
.
```

Symbole, Opcodes und Assembler-Anweisungen können sowohl in Groß- als auch Kleinbuchstaben notiert werden, der Assembler macht keine Unterscheidung:

LABEL = label = Label

### 2.3. Command-Line-Switches, Filenamen

Die Command-Line-Switches ermöglichen zum Zeitpunkt des Assembler-Stertes den Ort der Ziel-Dateien festzulegen.

Für jede der 3 Ziel-Dateien gibt es einen Switch:

```
P --> Switch für Listing      (.LST)
O --> Switch für Object-Code  (.COD)
E --> Switch für EPROM-Code   (.BIN)
```

Die Switches können jeweils in verschiedene 'Stellungen' gebracht werden:

```
N --> diese Zieldatei nicht erzeugen.
X --> diese Zieldatei auf den Bildschirm
      leiten (nur für Listing).
P --> diese Zieldatei an den Drucker
      leiten (nur für Listing).
A...M --> diese Zieldatei auf Drive A: ... M:
          leiten.
```

Jeder Switch wird durch einen Slesh (/) eingeführt.  
Zulässige Switches sind also:

```
A>DPAL HALLO/PO/EN
A>DPAL HALLO /PX
A>DPAL HALLO /PP /OB /EX
```

Eine bestimmte Reihenfolge braucht nicht eingehalten zu werden.

Werden keine Switches eingegeben, so gilt folgender Default:

Alle 3 Ziel-Dateien werden auf dem Drive angelegt, auf dem sich auch die Quell-Datei befindet.

#### Beispiele:

```
A>DPAL D:MUSTER --> Ziel-Dateien auf D:
A>B:DPAL MUSTER --> Ziel-Dateien auf A:
A>B:OPAL C:MUSTER --> Ziel-Dateien auf C:
```

Für OPAL-68000 Quell-Programme gilt folgende Namens-Festlegung:

Wird ein Quell-File ohne Namens-Extent angegeben, so wird automatisch der Extent:  
    '.M68'

verwendet. Ansonsten muß ein anderer Extent explizit angegeben werden.

#### Beispiele:

Command-Line:	Quell-Datei:
A>OPAL MUSTER	MUSTER.M68
A>OPAL MUSTER.M68	MUSTER.M68
A>OPAL MUSTER.123	MUSTER.123

#### V e r m e i d e n    S i e :

A>OPAL MUSTER.BIN    --> löscht Quell-Datei !!  
    ... etc.

## 2.4. Assembler-Anweisungen (Pseudo-OPcodes)

### 2.4.1. XLIST - Pseudo

Ohne Argumente, schaltet an dieser Stelle die Erzeugung von Listing-Zeilen aus.

### 2.4.2. LIST - Pseudo

Ohne Argumente, schaltet die Erzeugung von Listing-Zeilen wieder ein.

Beispiel:

```
XLIST
; diese Zeile erscheint nicht im Listing
LIST
; dieser Teil des Quell-Codes wird wieder gelistet
;
;
```

Default zu Beginn eines Assembler-Quellprogramms:

```
LIST      ist aktiv
```

### 2.4.3. PAGE - Pseudo

Mit oder ohne Argument.

PAGE ohne Argument erzeugt im Listing einen Vorschub auf den nächsten Seitenumfang.

Mit Angabe eines Argumentes wird die Länge eines Seiteninhaltes definiert, ein Seitenvorschub wird nicht hervorgerufen.

Beispiel:

```
PAGE    64
TOP      8
```

legt fest, daß das Listing in Seiten von 64 Zeilen Länge erzeugt wird und der Abstand zwischen zwei Seiten 8 Zeilen beträgt. Diese Angabe ist z.B. sinnvoll für Druckerpapier, das eine Seitenlänge von 72 Zeilen aufweist.

Der zulässige Wertebereich für Page ist 0..255.

Werte < 8 führen dazu, daß ein Listing ohne Seitenstruktur erzeugt wird.

Default:            68        Zeilen pro Seite

### 2.4.4. TOP - Pseudo

Mit einem Argument, legt die Anzahl der Zeilen zwischen 2 Listing-Seiten fest.

Wertebereich:    0 ... 255

Beispiel:        --> siehe 'PAGE'

Default:        4    Zeilen zwischen den Seiten

#### 2.4.5. LINE - Pseudo

Mit einem Argument, legt die maximale Anzahl der Zeichen pro Druckzeile im Listing fest. Enthält eine Druckzeile mehr Zeichen, als durch die LINE-Anweisung zugelassen, werden die restlichen Zeichen unterdrückt. Nur eine LINE-Anweisung im Quellcodes ist wirksam (die letzte) und hat Gültigkeit für das gesamte Listing.

Wertebereich: 79 ... 255

Beispiel: --> siehe Demo-Programm 'DTEST3.M68'  
S. 128 ff

Defaults: 79 Druck-Zeichen pro Zeile

#### 2.4.6. LINIT - Pseudo

Mit Argument vom Typ BYTE und/oder STRING.

LINIT definiert eine 'Listing-Initialisierungs-Sequenz', die zu Beginn des Listings generiert wird. Aufgabe einer solchen Sequenz ist die Selektierung bestimmter Drucker-Funktionen, z.B:

- Auswahl eines Zeichensatzes
- Einstellung der Zeichen-Breite (hor.-Pitch)
- Einstellung des Zeilenabstandes (vert.-Pitch)
- Reset-Funktion, Drucker-Status in definierte Ausgangslage bringen.

Beispiel:

```
ESC: EQU 27      ; Escape
EA:  EQU 58      ; eckige Klammer auf
```

; horiz. Pitch auf 1/12" stellen für LA-50 Drucker:

```
LINIT ESC,EA,'2w'
```

Nur eine LINIT-Anweisung (die letzte) ist wirksam. Ohne LINIT-Anweisung wird vor dem Listing kein weiteres Zeichen ausgegeben.

---> siehe auch Demo-Programm 'DTEST3.M68' S. 128 ff

#### 2.4.7. LEXIT - Pseudo

Mit Argument vom Typ BYTE und/oder STRING.

LEXIT definiert eine 'Listing-Exit-Sequenz', die am Ende des Listings generiert wird. Aufgabe einer solchen Sequenz ist es, nach einer Listing-Ausgabe auf dem Drucker, wieder den 'normalen' Betriebs-Zustand des Druckers herzustellen.

##### Beispiel:

Das Listing eines 68000-Programm soll auf einem Matrix-Drucker mit 95 Zeichen/Zeile ausgegeben werden. Des verwendete Drucker-Papier habe annähernd DIN-A4 Größe. Bei dem für Textverarbeitung häufig verwendeten Pitch von 1/10" sind nur etwa 80 Zeichen je Druckzeile möglich. Es wird daher in diesem Fall von der Anweisung LIMIT Gebrauch gemacht, mit der ein Pitch von 1/12" eingestellt wird (s.o.). Nach Abschluß des Listing-Ausdrucks wird jedoch wieder die ursprüngliche Einstellung von 1/10" Pitch gewünscht, dies ist mit der LEXIT-Anweisung möglich:

```
ESC: EQU 27 ; Escape  
EA: EQU 5B ; eckige Klammer auf
```

```
; horiz. Pitch auf 1/12" stellen für LA-50 Drucker:  
LIMIT ESC,EA,'24'
```

```
; nach Abschluß des Listings wieder 1/10"-Pitch:  
LEXIT ESC,EA,'0w'
```

Nur eine LEXIT-Anweisung (die letzte) ist wirksam. Ohne LEXIT-Anweisung wird nach dem Listing kein weiteres Zeichen ausgegeben.

## 2.4.8. XPUNCH - Pseudo

Ohne Argument, unterdrückt die Ausgabe des Maschinen-Codes im Listing ab der aktuellen Position.

Beispiel:

```
003042 446965736573      DC.B  'Dieses ist ein String 1',STOP,12,12x8,12x
003046 206973742065
00304E 696E20537472
003054 696E67202100
00305A 0C0A12
00305D 446965736573      DC.B  'Dieses ist ein String 1',STOP,12,12x8,12x
003063 206973742065
003069 696E20537472
00306F 696E67202100
003075 0C0A12
003078 446965736573      DC.B  'Dieses ist ein String 1',STOP,12,12x8,12x
00307E 206973742065
003084 696E20537472
00308A 696E67202100
003090 0C0A12

XPUNCH | Maschinen-Code nicht mehr ausdrucken |
003093      DC.B  'Dieses ist ein String 1',STOP,12,12x8,12x
0030A6      DC.B  'Dieses ist ein String 1',STOP,12,12x8,12x
0030C9      DC.B  'Dieses ist ein String 1',STOP,12,12x8,12x
0030E4      NOP
```

## 2.4.9. PUNCH - Pseudo

Ohne Argument, schaltet die Ausgabe des Maschinen-Codes im Listing ab der aktuellen Position ein.

Beispiel: ---> siehe 2.4.8. (XPUNCH)

Default-Wert zu Beginn eines Listings ist: Maschinen-Code wird ausgedruckt.



#### 2.4.10. TITLE - Pseudo

Mit Argument vom Typ BYTE und/oder STRING.  
Die TITLE-Anweisung erlaubt die individuelle  
Beschriftung der Seiten eines Listings.

Auf dem nächsten der TITLE-Anweisung folgenden  
Seitenkopf wird die definierte Überschrift  
abgedruckt.

Neben reinem Text sind auch die verschiedenen  
Drucker-Steuerzeichen erlaubt (Fettdruck,  
Unterstreichung, Farbwechsel, etc.)

Beispiel:

```
TITLE   ESC,EA,'6wTitel-Zeile',ESC,EA,'2w'
```

erzeugt in einem Listing (auf Drucker LA-50):

**Titel-Zeile**

Beliebig viele TITLE-Anweisungen sind zugelassen,  
ohne TITLE-Anweisung im Programm wird eine  
Leerzeile an entsprechender Stelle gedruckt.

Enthält die erste Zeile eines 68000-Quellprogramms  
eine TITLE-Anweisung, so wird der Titel bereits ab  
der ersten Listing-Seite ausgedruckt.

## 2.4.11. XFLAG - Pseudo

Ohne Argument, unterdrückt die Ausgabe der  
Flags im Listing ab der aktuellen Position.

Beispiel:

```

                                FLAG          ; Flags werden ausgedruckt
                                |-----|
000335 4E71          LAB_1: NOP
000337 68FC          BRA     LAB_1
000339 4E71          N  LAB_2: NOP
00033B 6400FFFF      R  LAB_2: BCC.W LAB_1
00033F 4E71          N  LAB_3: NOP

                                XFLAG         ; keine Flags mehr ausdrucken
                                |-----|
000561 4E71          LAB_4: NOP
000563 68FC          BRA     LAB_4
000565 4E71          LAB_5: NOP      ; N-Flag
000567 6400FFFF      LAB_5: BCC.W LAB_4 ; R-Flag
00056B 4E71          LAB_6: NOP      ; N-Flag
                                FLAG         ; Flags wieder drucken
                                |-----|

```

## 2.4.12. FLAG - Pseudo

Ohne Argument, schaltet die Ausgabe der  
Flags im Listing ab der aktuellen Position  
ein. (Beispiel: ---> siehe XFLAG (2.4.11.))

Beliebig viele XFLAG / FLAG - Anweisungen sind  
zulässig. Default-Wert zu Beginn des Listings ist:

Flags werden ausgedruckt.

Mögliche Flags:

- N = Nicht benutztes Symbol
- = Symbol-REDEF
- R = Range, es kann eine kleinere Objekt-Größe  
gewählt werden.
- C = File-INCLUDE ist aktiv

## 2.4.13. DC - Pseudo

Mit Argument(en), definiert Konstanten.

Mit der DC-Anweisung können Konstanten der Größe:

BYTE  
WORD  
LONG

definiert werden.

Als Argument-Typen werden akzeptiert:

numerische Konstanten,  
symbolische Konstanten,  
Strings,  
Labels,  
Programm-Counter  
sowie arithmetische Ausdrücke aus diesen.

Die Größe der durch DC definierten Objekte wird entweder:

explizit oder  
per default

festgelegt.

Beispiele:

```

0004A0 446965736573          ; SIZE explicit and by Default:
0004B0 206973742065          DC.W  'Dieses ist ein String ',STOP,12,12x8,12x
0004B6 696E20537472
0004BC 696E67202100
0004C2 0000C000A000
0004C8 12
0004C9 446965736573          DC.L  'Dieses ist ein String ',STOP,12,12x8,12x
0004CF 206973742065
0004D5 696E20537472
0004DB 696E67202100
0004E1 000000000000
0004E7 0C00000000A000
0004ED 000012

```

```

                                SIZE.L
0004F8 446965736573      DC  'Dieses ist ein String !',STOP,12,12x8,12H
0004F6 206973742065
0004FC 696E20537472
000502 696E67202100
000500 000000000000
00050E 0C0000000000
000514 000012

                                SIZE.W
000517 446965736572      DC  'Dieses ist ein String !',STOP,12,12x8,12H
000510 206973742065
000523 696E20537472
000529 696E67202100
00052F 0000C0000000
000535 12

                                SIZE.B
000536 446965736573      DC  'Dieses ist ein String !',STOP,12,12x8,12H
00053C 206972742065
000542 696E20537472
000548 696E67202100
00054E 0C0012

```

Zur Komprimierung des Listings bei größeren Text-Bereichen kann die Ausgabe des Maschinen-Codes mit der XPU\$CH-Anweisung unterdrückt werden. --> siehe 2.4.8.

## 2.4.14. DS - Pseudo

Mit einem Argument, definiert einen Speicher-Bereich. Die Objekt-Größen BYTE, WORD und LONG können sowohl explizit oder per Default vereinbart werden.

Beispiele:

```

; Objekt-Größe explizit
|
DS.B    79          ; erzeugt einen Block von 79 Bytes
DS.W    79          ; 79 Words = 158 Bytes
DS.L    12H         ; 12H Longs = 18 Longs
              ;      = 36 Words
              ;      = 72 Bytes
|
; Objekt-Größe per Default:
|
SIZE.W
DS      4           ; belegt 4 Words = 8 Bytes
SIZE.L
DS      VIELE       ; belegt VIELE Longs,
              ;      = 4*VIELE Bytes

```

Der von der DS-Anweisung generierte Block ist mit einem bestimmten Zeichen initialisiert. Default-Zeichen ist 00H. Mit der in 2.4.15. beschriebenen FILL-Anweisung kann ein beliebiges anderes Zeichen definiert werden.

## 2.4.15. FILL - Pseudo

Mit Argument, definiert ein Füllzeichen für die DS-Anweisung.

Beispiel:

```

FILL    'A'
DS.B    20          ; Block von 20-Bytes mit 41H
              ; initialisiert.

```

Zulässiger Wertebereich für das Argument:

0 ... 255

**2.4.16. EVEN - Pseudo**

Ohne Argument, stellt den Programm-Counter auf die nächste gerade Adresse.

Beispiele:

```
EVEN
DC      'Hallo'                                | PC ungerade
EVEN                                         | PC auf gerade ADR
Do_somethings:  BSR      Init_4
                  BCC.B   Part_5
                  .
                  .
```

Die EVEN-Anweisung wird in der Regel hinter einer DC oder DS-Anweisung benutzt, wenn nicht sicher ist, ob der PC noch auf einer geraden Adresse steht.

Sofern der PC tatsächlich auf einer ungeraden Adresse steht, wird ein Zeichen in den Objekt-Code eingefügt. (00H ist Default, sonst entsprechend der letzten FILL-Anweisung).

## 2.4.17. EQU - Pseudo

Mit einem Argument, weist einem Symbol einen Wert zu.

Zulässige Argumente sind:

- numerische Konstanten,
- symbolische Konstanten,
- Labels,
- String-Konstanten,
- Programm-Counter,
- arithmetische Ausdrücke hieraus.

Eine Vorwärts-Referenz, beliebig viele Rückwärts-Referenzen sind möglich.

Beispiele:

```

                                | verschiedene EQUs
43444346      N  ABCDEF: EQU   'ABCDEF'
42434443      N  ABCDE:  EQU   'ABCDE'
41424344      N  ABCD:   EQU   'ABCD'
00414243      N  ABC:    EQU   'ABC'
00004142      N  AB:     EQU   'AB'
00000041      N  A:      EQU   'A'
00000000      N  X:      EQU   ''

01046001      SYMBOL_1: EQU  011011010011010111010001X2  ; binär
FE25942F      N  SYMBOL_2: EQU  -01110110100110101111010001X2  ; binär
3A3F3801      N  SYMBOL_3: EQU  76543217654321XB  ; octal
B7B6B5B5      N  SYMBOL_4: EQU  ~'ABCDEFQHIJK'  ; ASCII
49968202      N  SYMBOL_5: EQU  1234567890  ; decimal
0345440C      N  SYMBOL_6: EQU  045440C  ; hexa
12345678      SYMBOL_7: EQU  12345678H  ; hexa
12345678      N  SYMBOL_8: EQU  SYMBOL_9
12345678      SYMBOL_9: EQU  SYMBOL_7
00414325      N  SYMBOL_10: EQU  $  ; PC

```

In der EQU-Anweisung dürfen nur solche Symbole definiert werden, die noch nicht existieren. Re-Definition ist ggf. mit der REDEF-Anweisung durchzuführen.

## 2.4.18. ORG - Pseudo

Mit einem Argument, setzt den Programm-Counter.

Zu Beginn eines 68000 Programms ist mindestens eine ORG-Anweisung vorgeschrieben. Im Programm-Verlauf können weitere folgen.

Beispiel:

```

|      Name:  DEMO.M68
|      Typ:   OFAL-68000 Source
|      Stand: 6.6.84 (1)
|
ESC:      EQU      27
BELL:     EQU      7
BASE_ADR: EQU      2000H
BASE_2:   EQU      1234H
|
ORG      BASE_ADR
NOP
NOP

ORG      BASE_2
NOP
NOP

```

## 2.4.19. SIZE - Pseudo

Ohne Argument, mit SIZE-Extent, setzt die jeweils gültige Default-SIZE.

Zulässige SIZE-Extents:

.B	Byte
.W	Word
.L	Long
.S	Small = Byte

Beispiele:

per Default:		explizit:	
SIZE.B			
BRA LABEL	(==)	BRA.B LABEL	
SIZE.L			
DS 4	(==)	DS.L 4	



#### 2.4.20. PRINT - Pseudo

Mit Argument vom Typ Byte und/oder String.

Das Argument von PRINT wird im Fass-1 als Meldung an die Console ausgegeben. Solche Meldungen können benutzt werden um den Assembler-Vorgang von Programmen zu verfolgen, oder im Zusammenhang mit der Input-Anweisung zur interaktiven Assemblierung. (siehe 2.4.21)

##### Beispiel:

```
.  
.  
BELL: EQU 7 ; Glocke  
CR: EQU 13  
LF: EQU 10  
  
PRINT BELL, Programm-Teil 1',CR,LF  
.  
.
```

## 2.4.21. INPUT - Pseudo

Ohne Argument, weist einem Symbol während Pass-1 des Assemblerlaufs interaktiv einen Wert zu.

Beispiele:

```
PRINT  BELL, 'Start-Adresse = '
START: INPUT                ; Adr vom user holen
PRINT  CR, LF               ; neue Zeile

      ORG      START
MAIN:  ...
```

Die INPUT-Anweisung arbeitet ähnlich der EQU-Anweisung indem sie einem Symbol einen Wert zuweist, jedoch wird der Wert nicht im Assembler-Quellprogramm festgeschrieben, sondern erst zum Zeitpunkt des Assembler-Laufs vom Bediener eingegeben. Da die INPUT-Anweisung keinerlei Zeichen an den Bildschirm ausgibt (wie das mitunter beim Basic 'INPUT' ist), sollte zuvor mit der PRINT-Anweisung ein geeigneter Hinweis ausgegeben werden (s.o.).

Die Anwendung der INPUT-Anweisung ist stets denn vorteilhaft, wenn häufig veränderte Parameter benötigt werden, also z.B. in der Entwicklungs- und Test-Phase eines Programms.

Eine weitere Anwendung ist z.B. die kontrollierte Listing-Erzeugung. Bei einem längeren Programm soll nicht jedesmal ein komplettes Listing ausgedruckt werden, sondern nur der eine oder andere Teil. Um für diese Aufgabe möglichst flexibel zu bleiben, ist folgende Konstruktion geeignet:

```
| Name:  INTERAKT.M68
| Typ:   DPAL-68000 Source
| Stand: 16.7.84 (2)
| Funktion: Kontrollierte Erzeugung von Gesamt-/Teil-Listings
```

```
ON: EQU OFFH ; 'ein'
OFF: EQU 0    ; 'aus'
J: EQU ON     ; 'ju'
N: EQU OFF    ; 'nein'
BELL: EQU 7   ; Glocke
CR: EQU 13
LF: EQU 10
STOP: EQU 0
```

```

MODUL_1_START: EQU    1000H
MODUL_2_START: EQU    14000H

PRINT  CR,LF,'Listing-Erzeugung:',CR,LF
PRINT  '=====',CR,LF,LF,BELL
PRINT  'Gesamt-Listing ?? (J/N) '
LISTING: INPUT          ; hole Antwort vom user

PRINT  CR,LF
IFN    LISTING ; ==> Gesamt-Listing erzeugen
TEIL_1: EQU    J      ; ==> alle Teil-Listings setzen !
TEIL_2: EQU    J
TEIL_3: EQU    J
TEIL_4: EQU    J
ENDIF

    IF    LISTING
PRINT  'Teil-1 listen ?? (J/N) '
TEIL_1: INPUT

PRINT  CR,LF,'Teil-2 listen ?? (J/N) '
TEIL_2: INPUT

PRINT  CR,LF,'Teil-3 listen ?? (J/N) '
TEIL_3: INPUT

PRINT  CR,LF,'Teil-4 listen ?? (J/N) '
TEIL_4: INPUT

PRINT  CR,LF
ENDIF

    IF    TEIL_1 ; Listing Teil-1
XLIST
ENDIF

ORG    MODUL_1_START          ; T e i l 1
MAIN:  BNA    ANFANG_1        ; =====
      SIZE.B
      DC     CR,LF,'XYZ-Software vers. x.xx'
      DC     CR,LF,'Copyright 1984 Mustermann',CR,LF,STOP

      LIST
      IF    TEIL_2 ; Listing Teil-2
XLIST
      ENDIF

ANFANG_1: NOP                ; T e i l 2
        NOP                ; =====
        NOP
        NOP
        NOP

```

```

LIST
IFE      TEIL_3 ; Listing Teil-3
XLIST
ENDIF

      BRA      ANFANG_1      ; Teil 3
LABEL1: NOP
LABEL2: NOP
LABEL3: NOP
LABEL4: NOP
LABEL5: NOP
LABEL6: NOP
LABEL7: NOP
LABEL8: NOP

      LIST
IFE      TEIL_4 ; Listing Teil-4 (Symbol-Tabelle)
XLIST
ENDIF

; Teil 4 = Symbol-Tabelle
; ~~~~~

```

Zulässige Objekt-Größe = LONG,  
falsche Eingaben, oder keine Eingabe erzeugen  
den Symbol-Wert 0.

Als zulässige Eingaben sind alle erlaubten  
Argumente der EQU-Anweisung anzusehen, also:

- numerische Konstanten,
- symbolische Konstanten, (\*)
- Labels, (\*)
- String-Konstanten,
- Programm-Counter,
- arithmetische Ausdrücke hieraus.

(\*) --> müssen im Pass-1 an dieser Stelle  
bereits bekannt sein, da sonst der  
Wert 0 für das unbekannte Symbol  
genommen wird.

## 2.4.22. IFE, IFN, IFP, IFM - Pseudos

Mit einem Argument, leitet die bedingte Assemblierung ein.

Falls nicht schon eine bedingte Assembler-Abschaltung aktiv ist, wird der Wert des Argumentes getestet. Geht der Test positiv aus, so wird der folgende Programmteil assembliert, andernfalls wird die Assemblierung ab dieser Anweisung ausgeschaltet.

Zu jeder IFx-Anweisung gehört eine ENDIF-Anweisung, die die Wirkung der IFx-Anweisung beendet.

Die IFx-Anweisungen können bis zu einer Tiefe von 254 verschachtelt werden.

Je nach verwendeter Anweisung können folgende Bedingungen unterschieden werden:

```
IFE  -->  erfüllt wenn Arg  = 0
IFN  -->  erfüllt wenn Arg  <> 0
IFP  -->  erfüllt wenn Arg  >= 0
IFM  -->  erfüllt wenn Arg  < 0
```

## Beispiele:

```

; Test Conditional-Assembly, Bedingung ist erfüllt
; *****
00056F 4E71      IFE    5 + 6 - 11
                NOP
                ENDF

000571 4E71      IFN    9876 + SYMBOL
                NOP
                ENDF

000573 4E71      IFP    8352
                NOP
                ENDF

000575 4E71      IFM    -5555
                NOP
                ENDF

```

```
; Bedingung nicht erfuehlt:
;=====
IFB      01910111XZ
NOP
ENDIF

IFM      0000GX0
NOP
ENDIF

IFP      -1234567X0
NOP
ENDIF

IFM      SSSS
NOP
ENDIF


; verschachtelt:           - Schachtelungsstiefe: -
IFB      0                    ; 1
NOP
IFM      1                    ; 1
NOP                                ; 2
IFP      2                    ; 3
NOP                                ; 3
IFM     -3                    ; 4
NOP                                ; 4
IFE      0                    ; 5
NOP                                ; 5
IFM      0                    ; Bedingung nicht erfuehlt
NOP                                ; 6
IFP      3                    ; 7
NOP                                ; 7
IFM     -87                   ; 8
NOP                                ; 8
IFE      0                    ; 9
NOP                                ; 9
ENDIF
NOP                                ; 8
ENDIF
NOP                                ; 7
ENDIF
NOP                                ; 6
ENDIF
NOP                                ; 5
ENDIF
NOP                                ; 4
ENDIF
NOP                                ; 3
ENDIF
NOP                                ; 2
ENDIF
NOP                                ; 1
ENDIF
```

2.4.23.      **ENDIF - Pseudo**

Ohne Argument, schließt eine IFx-Anweisung ab. - Siehe 2.4.22.

2.4.24.      **INCLUDE - Pseudo**

Mit einem Argument vom Typ Byte und/oder String. Schließt den im Argumententeil spezifizierten Disk-File in den Quell-File ein.

Beispiele:

```
Drive: EQU 'C'
INCLUDE 'A:TEIL-2.M68'
INCLUDE Drive,'Teil-3.M68'
```

INCLUDE-Anweisungen können nur im Hauptfile ausgeführt werden, aus INCLUDE-Files heraus können keine weiteren Files eingeschlossen werden.

Listing-Zeilen aus INCLUDE-Files werden im Flag-Feld durch des 'C'-Flag gekennzeichnet.

2.4.25.      **REDEF - Pseudo**

Mit einem Argument, definiert ein bereit vorhandenes Symbol neu.

Beispiele:

```

01046801      SYMBOL_1:EQU 01110110100110101111010001X2    ; binär
00000001      -    SYMBOL_1:REDEF 1
414320 063C0001      ADD    #SYMBOL_1,03
00000002      -    SYMBOL_1:REDEF 2
414330 063C0002      ADD    #SYMBOL_1,03
00000003      -    SYMBOL_1:REDEF 3
414335 063C0003      ADD    #SYMBOL_1,03
```

## 2.5. Adressierungsarten

ADR-Art:	Beispiel:
Dn	ADD.L D3,D6
An	ADD.W A2,D6
(An)	ADD.B (A1),D6
(An)+	ADD.B (A3)+,D6
-(An)	ADD.B -(A4),D6
d(An)	ADD.B DFFSET (A5), D6
d(An,Ri)	ADD.B DFFSET (A1,D2.W),D6
Abs.W	ADD.B @ LABEL.W,D6
Abs.L	ADD.B @ LABEL.L,D6
	ADD.B @LABEL,D6
d(PC)	ADD DISPLACEMENT (\$),D6
d(PC,Ri)	ADD SMALL_DISPL (\$, D2.L),D6
Imm	ADD.L @ [VIEL-3+K7] * SYMBOL,D6

CCR - Condition-Code Register (Flags):

ANDI.B %data,CCR

SR - Status-Register:

ANDI.W %XYZ,SR

USP - User Stackpointer:

MOVE.L USP,A3

Register-Liste:

MOVEH A2/A1/D1/D6/D3 , 12H(A4)

Register werden durch einen Buchstaben (A bzw. D) und mit einer bündigen Ziffer (0...7) dargestellt.

Bündig auszuschreiben sind auch:

(An)  
(An)+  
-(An)  
(An,Ri)  
(\$)  
(\$,Ri)

sowie alle SIZE-Angebaben.



Nicht bündig geschrieben werden brauchen:

Offsets,

•

•

Ausdrücke

## 2.6. Symbole, Konstanten, Operatoren, Ausdrücke

Symbole werden auf einer Länge von 12 Zeichen unterschieden, können jedoch auch länger sein.

Jedes Symbol muß mit einem Buchstaben beginnen, nachfolgend können zusätzlich Ziffern und das Underline-Zeichen verwendet werden. Ein Symbol ist bei seiner Definition durch einen bundigen Doppelpunkt abzuschließen.

Beispiele:

```
SYMBOL_1:    EQU    1234567890
SYMBOL_2:    INPUT
SYMBOL_1:    PEDEF  7654321
LABEL_1:     NOP
```

Groß- und Kleinbuchstaben werden nicht unterschieden:

```
LABEL = label = Label
```

Konstanten können in verschiedenen Zahlensystemen und als String-Ausdruck angegeben werden:

```
String-Definition:  '...'      'String'
Zahlenbasis 2:      'X2'        1010001111X2
Zahlenbasis 8:      'XB'        77212601XB
Zahlenbasis 10:     -           9124
Zahlenbasis 16:     'H'         0A4FF5BDH
                        oder:     0A4FF5BD
```

Zahlenkonstanten beginnen stets mit einer Ziffer, ggf. also mit einer Null: 0ABCDH

Konstanten und Symbole können durch Operatoren zu arithmetischen Ausdrücken verknüpft werden.

Zulässige Operatoren sind:

```
* / ?      Multiplikation, Division, Rest
- +        Subtraktion, Addition
! & %      XOR, AND, OR
```

(nach absteigender Priorität geordnet)

Alle Operationen werden in vorzeichenloser 32-bit Integer-Arithmetik ausgeführt, negative Zahlen werden in 32-bit 2-er Komplement-Darstellung abgelegt. Die Bearbeitung von Ausdrücken erfolgt von links nach rechts unter Berücksichtigung der angegebenen Prioritäten. Durch Klammerung mit eckigen Klammern ([...]) werden Prioritäten entsprechend verändert, Überläufe über die höchste Stelle hinaus werden ignoriert.

Bei der Verwendung von String-Ausdrücken ist folgende Unterscheidung zu beachten:

'abcdefghi' kann als STRING oder als ZAHL benutzt werden:

```

      DC.B  'abcdefghi'      ; hier: String !
VAR_11 EQU  'abcdefghi'    ; hier: Zahlenwert !

```

In diesen beiden Beispielen ist die Verwendung des Stringausdrucks als STRING bzw. als ZAHL eindeutig. Bei anderer Gelegenheit bedarf es einer entsprechenden Festlegung in der Programmzeile:

```

DC.B  'abcdefghi'      ; OK, 9 ASCII-Zeichen (String)
DC.L  'abcdefghi' / 4 + 7 ; Fehler !!
DC.L  ['abcdefghi'] / 4 + 7 ; OK, arithm. Ausdr. (Zahl)

```

Bei allen Assembler-Anweisungen die Strings zulassen, ist deshalb eine entsprechende Kennzeichnung erforderlich, falls ein String als Zahl verwendet werden soll:

```

z.B:   0 + 'AB'
        1 * 'abcd' - 'W' + 'OP'
        [ 'ABCOE' ]

```

Beispiele:

```

001000 000000DC      DC.L  [(34 + 'A' + A + 0A) + 'B' - 5] / SYMBOL
001004 646263736573  DC.L  'dieses ist ein String'      ; String-Interpretation
00100A 206973742065
001010 696E20537472
001016 696E67
001019 0177A9A2      DC.L  ['dies' / 67]                ; beachte Klammerung !

0000007E           A1 EQU 126
00000003           SYMBOL EQU 3

```

## 2.7. Fehlermeldungen

Fehler in Sourcecode-Zeilen werden durch Buchstaben zu Beginn der Listing-Zeile gekennzeichnet.  
Bis zu 3 Fehlern werden pro Zeile angezeigt.

- A = Argument-Fehler, Argument ist fehlerhaft, zu viele oder zu wenige Argumente, unzul. ADR-Mode
- I = Illegales Zeichen in der Zeile (z.B. CTRL-Character)
- L = LABEL-Fehler
- M = Mehrfache Symbol-Definition, bzw. Verwendung eines solchen Symbols
- N = Numerischer Fehler
- O = Opcode-Fehler
- P = Phasen-Fehler, z.B. mehr ENOIF-Anweisungen als IFx oder REDEF vor EQU
- R = Bereichs-Überschreitung eines Zahlenwertes
- S = SIZE-Fehler, eine SIZE-Angebe in dieser Zeile ist unkorrekt.
- U = undefiniertes SYMBOL wird verwendet
- X = sonst. Fehler

ABCD	Dezimal Addition	ABCD
------	------------------	------

Notation:      ABCD      Dx,Dy  
                  ABCD      -(Ax),-(Ay)

Objekt-Größe:    Byte

Funktion:        (op1) + (op2) + X    ---> (op2)

Addiert Operand-1 und Operand-2 sowie das X-Flag, nach der Operation enthält Operand-2 die Summe, das X-Flag den Übertrag. Die Addition geht von der dezimalen Zahlendarstellung im BCD-Code aus:

Beispiel:    (4-stellig)

	6659	0110 0110 0101 1001
	+ 5267	+ 0101 0010 0110 0111
Überträge:	1   11	1   0   1   1   0
	<hr/>	<hr/>
Übertr./Summe	/1/ 1926	/1/ 0001 1001 0010 0110

Je ein Byte enthält 2 BCD-Ziffern:

bit-Nr:	7 6 5 4	3 2 1 0
	Zehner-	Einer-
	Stelle	Stelle

#### Flags:

X - Gesetzt falls dezimaler Überlauf, sonst rückgesetzt.  
 C - Ofo.  
 V - Nicht definiert.  
 Z - Rückgesetzt falls Ergebnis (<) 0, sonst unverändert.  
 N - Nicht definiert.

---

ADD	Binär Addition	ADD
-----	----------------	-----

---

Notation:      ADD      <ea> , Dn  
                  ADD      Dn , <ea>

Objekt-Größe:    Byte, Word, Long

Funktion:        (op1) + (op2)    ---> (op2)

Addiert binäre Objekte der Größe Byte, Word oder Long.  
 Operand-2 enthält nach der Operation das Ergebnis.

### Flags:

- X - Gesetzt falls Übertrag erzeugt wird, sonst rückgesetzt.
- C - Dto.
- V - Gesetzt falls Überlauf erzeugt wird, sonst rückgesetzt.
- Z - Gesetzt falls Ergebnis = 0, sonst rückgesetzt.
- N - Gesetzt falls Ergebnis negativ, sonst gesetzt.

Adressierungsarten für Operand-1: <ea>

Dn	(An)+	d(An,Ri)	d(\$)
An	-(An)	Abs.W	d(\$,Ri)
(An)	d(An)	Abs.L	Imm

Adressierungsarten für Operand-2: <ea>

.	(An)+	d(An,Ri)	.
.	-(An)	Abs.W	.
(An)	d(An)	Abs.L	.

Objekt-Größen für Address-Register direkt: Word und Long

---

ADDA	Adress Addition	ADDA
------	-----------------	------

---

Notation:           ADDA           <ea> , An

Objekt-Größe:   Word, Long

Funktion:           (op1) + (op2)   ---> (op2)

Addiert Operand-1 und Operand-2 binär, Ergebnis wird im Adress-Register An abgelegt.

#### Flags:

X - Unverändert.  
C - Unverändert.  
V - Unverändert.  
Z - Unverändert.  
N - Unverändert.

Adressierungsarten für Operand-1: <ea>

Dn	(An)+	d(An,Ri)	d(\$)
An	-(An)	Abs.W	d(\$,Ri)
(An)	d(An)	Abs.L	Imm

---

<b>ADDI</b>	Addition mit Konstante	<b>ADDI</b>
-------------	------------------------	-------------

---

Notation:            ADDI #<datum>,<ee>

Objekt-Größe:    Byte, Word, Long

Funktion:            (op1) + (op2)    --->    (op2)

Addiert Operand-1 und Operand-2 binär, des Ergebnis wird nach Operand-2 geschrieben.

#### Flags:

- X - Gesetzt falls Übertrag erzeugt wird, sonst rückgesetzt.
- C - Dto.
- V - Gesetzt falls Überlauf erzeugt wird, sonst rückgesetzt.
- Z - Gesetzt falls Ergebnis = 0, sonst rückgesetzt.
- N - Gesetzt falls Ergebnis negativ, sonst rückgesetzt.

Adressierungsarten für Operand-2: <ea>

Dn	(An)+	d(An,R1)	.
.	-(An)	Abs.W	.
(An)	d(An)	Abs.L	.



---

ADDQ	Addiere schnell	ADDQ
------	-----------------	------

---

Notation:            ADDQ    #(<datum>),(<ee>)

Objekt-Größen:    Byte, Word, Long

Funktion:            (op1) + (op2)    --->    (op2)

Addiert Operand-1 und Operand-2 binär, das Ergebnis wird nach Operand-2 geschrieben.

Zulässiger Werte-Bereich für Operand-1:

000 ... 111 (binär)

### Flags:

- X - Gesetzt falls Übertrag erzeugt wird, sonst rückgesetzt.
- C - Ofo.
- V - Gesetzt falls Überlauf erzeugt wird, sonst rückgesetzt.
- Z - Gesetzt falls Ergebnis = 0, sonst rückgesetzt.
- N - Gesetzt falls Ergebnis negativ, sonst rückgesetzt.

Adressierungsorten für Operand-2: <ea>

On	(An)+	d(An,R1)	.
An	-(An)	Abs.W	.
(An)	d(An)	Abs.L	.

Objekt-Größen für Adress-Register direkt: Word und Long

---

ADDX	Addition mit X-Flag	ADDX
------	---------------------	------

---

Notation:      ADDX      Dx,Dy  
                 ADDX      -(Ax),-(Ay)

Objekt-Größe:    Byte, Word, Long

Funktion:            (op1) + (op2) + X ----> (op2)

Addiert Operand-1 und Operand-2 sowie des X-Flag binär, des Ergebnis wird nach Operand-2 geschrieben.

Flags:

- X - Gesetzt falls Übertrag erzeugt wird, sonst rückgesetzt.
- C - Dto.
- V - Gesetzt falls Überlauf erzeugt wird, sonst rückgesetzt.
- Z - Gesetzt falls Ergebnis = 0, sonst rückgesetzt.
- N - Gesetzt falls Ergebnis negativ, sonst rückgesetzt.

---

AND	UND - Funktion	AND
-----	----------------	-----

---

Notation:      AND      <ea>,Dn  
                  AND      Dn,<ea>

Objekt-Größe:    Byte, Word, Long

Funktion:        (op1) UND (op2)    ---> (op2)

Verknüpft Operand-1 und Operand-2 durch die UND-Funktion, das Ergebnis wird nach Operand-2 geschrieben.

#### Flags:

- X - Unverändert.
- C - Rückgesetzt.
- V - Rückgesetzt.
- Z - Gesetzt falls Ergebnis = 0, sonst rückgesetzt.
- N - Wird mit dem höchstwertigen bit des Ergebnisses geladen.

Adressierungsarten für Operand-1: <ea>

Dn	(An)+	d(An,Ri)	d(\$)
.	-(An)	Abs.W	d(\$,Ri)
(An)	d(An)	Abs.L	Imm

Adressierungsarten für Operand-2: <ea>

.	(An)+	d(An,Ri)	.
.	-(An)	Abs.W	.
(An)	d(An)	Abs.L	.

---

ANDI	UND-Funktion mit Konstante	ANDI
------	----------------------------	------

---

Notation:      ANDI      #<datum>,<ea>

Objekt-Größe:    Byte, Word, Long

Funktion:        (op1) UND (op2)    ---> (op2)

Verknüpft Operand-1 und Operand-2 durch die UND-Funktion, das Ergebnis wird nach Operand-2 geschrieben.

Zulässiger Werte-Bereich für Operand-1:

entsprechend der  
gewählten Objekt-Größe:    Byte, Word, Long

### Flags:

- X - Unverändert.
- C - Rückgesetzt.
- V - Rückgesetzt.
- Z - Gesetzt falls Ergebnis = 0, sonst rückgesetzt.
- N - Wird mit dem höchstwertigen bit des Ergebnisses geladen.

Adressierungsarten für Operand-2: <ea>

Dn	(An)+	d(An,Ri)	.
.	-(An)	Abs.W	.
(An)	d(An)	Abs.L	.

---

ANDI  
to CCR

UND mit Konstante nach CCR

ANDI  
to CCR

---

Notation:      ANDI      #(<datum>),CCR

Objekt-Größe:    Byte

Funktion:        (op1) + (op2)    ---> (op2)

Verknüpft den konstanten Operand-1 mit den dem Inhalt des Condition-Code-Registers. Das Ergebnis wird in des CCR geschrieben.

Zulässiger Werte-Bereich für Operand-1:

Byte

Flags:

- X - Rückgesetzt falls bit-4 des konstanten Operanden = 0, sonst unverändert.
- C - Rückgesetzt falls bit-0 des konstanten Operanden = 0, sonst unverändert.
- V - Rückgesetzt falls bit-1 des konstanten Operanden = 0, sonst unverändert.
- Z - Rückgesetzt falls bit-2 des konstanten Operanden = 0, sonst unverändert.
- N - Rückgesetzt falls bit-3 des konstanten Operanden = 0, sonst unverändert.

---

ANDI  
to SR

AND mit Konstante nach SR  
- privilegierter Befehl -

ANDI  
to SR

---

Notation: ANDI #<datum>,SR

Objekt-Größe: Word

Funktion: Falls 'Supervisor-State' aktiv:  
(op1) AND (op2) ---> (op2)  
andernfalls:  
führe TRAP aus

Verknüpft den konstanten Operand-1 mit den dem Inhalt des Status-Registers. Das Ergebnis wird in das Status-Register geschrieben.

Zulässiger Werte-Bereich für Operand-1:

Word

#### Flags:

- X - Rückgesetzt falls bit-4 des konstanten Operanden = 0, sonst unverändert.
- C - Rückgesetzt falls bit-0 des konstanten Operanden = 0, sonst unverändert.
- V - Rückgesetzt falls bit-1 des konstanten Operanden = 0, sonst unverändert.
- Z - Rückgesetzt falls bit-2 des konstanten Operanden = 0, sonst unverändert.
- N - Rückgesetzt falls bit-3 des konstanten Operanden = 0, sonst unverändert.

ASL	Arithm. links schieben	ASL
-----	------------------------	-----

Notation:

1.) ASL	Dx,Dy
2.) ASL	#<datum>,Dy
3.) ASL	<ea>

Objekt-Größe:

1.)	Byte, Word, Long
2.)	Byte, Word, Long
3.)	Word

Funktion: Ziel-Objekt um nn-Stellen links schieben

Das Ziel-Objekt (Operand-2 in den Fällen 1. und 2., Operand-1 im Fall 3.) wird um eine Anzahl (nn) bits arithmetisch nach links geschoben. Die Anzahl (nn) der Schiebeoperation ergibt sich zu:

- 1.) Anzahl steht im Register Dx, (modulo 64)
- 2.) Anzahl wird durch den unmittelbaren Operanden angegeben, mögliche Werte: 1..8
- 3.) Anzahl = 1

```

-----
1 C : (---
-----
!
!<-----;   Ziel - Objekt   !<-----: 0 1
!
!
! X : (---
-----

```

#### Flags:

- X - Wird mit dem zuletzt herausgeschobenen bit geladen.
- C - Wird mit dem zuletzt herausgeschobenen bit geladen, wird rückgesetzt, falls Anzahl der Schiebeoperationen = 0.
- V - Gesetzt falls das höchstwertige bit während der Ausführung der Schiebeoperation mindestens einmal verändert wurde.
- Z - Gesetzt falls das Ergebnis = 0, sonst rückgesetzt.
- N - Wird mit dem höchstwertigen bit des Ergebnisses geladen.

Adressierungsarten für Operand-1: <ea>

.	(An)+	d(An,R1)	.
.	-(An)	Abs.W	.
(An)	d(An)	Abs.L	.

ASR	Arithm. rechts Schieben	ASR
-----	-------------------------	-----

Notation:

- 1.) ASR      Dx,Dy
- 2.) ASR      \*(datum),Dy
- 3.) ASR      <ea>

Objekt-Größe:

- 1.) Byte, Word, Long
- 2.) Byte, Word, Long
- 3.) Word

Funktion:

Ziel-Objekt um nn-Stellen rechts schieben

Das Ziel-Objekt (Operand-2 in den Fällen 1. und 2., Operand-1 im Fall 3.) wird um eine Anzahl (nn) bits arithmetisch nach rechts geschoben. Die Anzahl (nn) der Schiebeoperation ergibt sich zu:

- 1.) Anzahl steht im Register Dx, (modulo 64)
- 2.) Anzahl wird durch den unmittelbaren Operanden angegeben, mögliche Werte: 1..8
- 3.) Anzahl = 1

```

                                     |----> C
-----|
|----> Ziel - Objekt |---->
|-----|
|         |         |----> X
|-----|

```

Flags:

- X - Wird mit dem zuletzt herausgeschobenen bit geladen.
- C - Wird mit dem zuletzt herausgeschobenen bit geladen, wird rückgesetzt, falls Anzahl der Schiebeoperationen = 0.
- V - Gesetzt falls das höchstwertige bit während der Ausführung der Schiebeoperation mindestens einmal verändert wurde.
- Z - Gesetzt falls das Ergebnis = 0, sonst rückgesetzt.
- N - Wird mit dem höchstwertigen bit des Ergebnisses geladen.

Adressierungsarten für Operand-1: <ea>

.	(An)+	d(An,Ri)	.
.	-(An)	Abs.W	.
(An)	d(An)	Abs.L	.



---

<b>Bcc</b>	<b>Bedingte Programmverzweigung</b>	<b>Bcc</b>
------------	-------------------------------------	------------

---

Notations:      Bcc      (Sprungziel)

Objekt-Größen:    Byte, Word

Funktion:          Falls Bedingung cc erfüllt:  
PC + Sprungziel-Distanz ---> PC

Bcc ist eine bedingte Programmverzweigung. 'cc' steht für eine der nachstehenden 14 Bedingungen:

Kurz:	Bedeutung:	Code:	logische Gleichung:
HI	- High	0010	$\bar{C} * \bar{Z}$
LS	- Low or same	0011	$C + Z$
CC	- Carry clear	0100	$\bar{C}$
CS	- Carry set	0101	$C$
NE	- Not Equal	0110	$\bar{Z}$
EQ	- Equal	0111	$Z$
VC	- Overflow clear	1000	$\bar{V}$
VS	- Overflow set	1001	$V$
PL	- Plus	1010	$\bar{N}$
MI	- Minus	1011	$N$
GE	- Greeter or equal	1100	$N * V + \bar{N} * \bar{V}$
LT	- Less then	1101	$N * \bar{V} + \bar{N} * V$
GT	- Greeter then	1110	$N * V * \bar{Z} + \bar{N} * \bar{V} * \bar{Z}$
LE	- Less or equal	1111	$Z + N * \bar{V} + \bar{N} * V$

---

**Bcc****Bedingte Programmverzweigung****Bcc**

---

- Fortsetzung -

Ein Sprung wird nur bei erfüllter Bedingung ausgeführt. Die Angabe des Sprungziels erfolgt relativ zum aktuellen PC-Stand. Zulässige Werte für die Sprungziel-Distanz sind:

- Byte: vorzeichenbehaftete ganze Zahl (8-bit),  
( -128 ... -1, 1 ... +127 )  
beachte: 0 ist nicht enthalten!
- Word: vorzeichenbehaftete ganze Zahl (16-bit),  
( -32768 ... 0 ... +32767 )

Sprünge auf die unmittelbar folgende Adresse sind nur mit einer Distanz-Angabe in der Größe 'Word' möglich.

#### Flags:

X - Unverändert  
C - Unverändert  
V - Unverändert  
Z - Unverändert  
N - Unverändert

**BCHG****Bit testen und verändern****BCHG****Notation:**

- 1.) BCHG Dn,<ea>
- 2.) BCHG #<datum>,<ea>

**Objekt-Größe:**

Byte für Objekte im Speicher,  
Long für Objekte in Daten-Register.

**Funktion:**

s.u.

BCHG testet ein bit von Operand-2 und setzt entsprechend dem Ergebnis das Z-bit auf 0 oder 1. Das gewählte bit im Operand-2 wird invertiert. Ist Operand-2 ein Daten-Register, so erfolgt die Angabe der bit-Nummer modulo-32, andernfalls modulo-8.

**Flags:**

- X - Unverändert.
- C - Unverändert.
- V - Unverändert.
- Z - Gesetzt, falls das getestete bit = 0, sonst rückgesetzt.
- N - Unverändert.

**1.) Adressierungsarten für Operand-2: <ea>**

Dn	(An)+	d(An,R1)	.
.	-(An)	Abs.W	.
(An)	d(An)	Abs.L	.

**2.) Adressierungsarten für Operand-2: <ea>**

Dn	(An)+	d(An,R1)	.
.	-(An)	Abs.W	.
(An)	d(An)	Abs.L	.

---

BCLR	Bit testen und rücksetzen	BCLR
------	---------------------------	------

---

Notation:

1.)	BCLR	Dn,<ea>
2.)	BCLR	*(<datum>),<ea>

Objekt-Größe: Byte für Objekte im Speicher,  
Long für Objekte in Daten-Register.

Funktion: s.u.

BCLR testet ein bit von Operand-2 und setzt entsprechend dem Ergebnis das Z-bit auf 0 oder 1. Das gewählte bit im Operand-2 wird rückgesetzt. Ist Operand-2 ein Daten-Register, so erfolgt die Angabe der bit-Nummer modulo-32, andernfalls modulo-8.

### Flags:

X - Unverändert.  
C - Unverändert.  
V - Unverändert.  
Z - Gesetzt, falls das getestete bit = 0, sonst rückgesetzt.  
N - Unverändert.

#### 1.) Adressierungsarten für Operand-2: <ea>

Dn	(An)+	d(An,R1)	.
.	-(An)	Abs.W	.
(An)	d(An)	Abs.L	.

#### 2.) Adressierungsarten für Operand-2: <ee>

Dn	(An)+	d(An,R1)	.
.	-(An)	Abs.W	.
(An)	d(An)	Abs.L	.

---

BRA	unbedingte Programmverzweigung	BRA
-----	--------------------------------	-----

---

Notation:        BRA        (Sprungziel)

Objekt-Größe:    Byte, Word

Funktion:        PC + Sprungziel-Distanz    --->    PC

BRA führt einen unbedingten Sprung aus. Die Angabe des Sprungziels erfolgt relativ zum aktuellen PC-Stand. Zulässige Werte für die Sprungziel-Distanz sind:

- Byte: vorzeichenbehaftete ganze Zahl (8-bit),  
      ( -128 ... -1, 1 ... +127 )  
      beachte! 0 ist nicht enthalten !
- Word: vorzeichenbehaftete ganze Zahl (16-bit),  
      ( -32768 ... 0 ... +32767 )

Sprünge auf die unmittelbar folgende Adresse sind nur mit einer Distanz-Angabe in der Größe 'Word' möglich.

#### Flags:

X - Unverändert  
C - Unverändert  
V - Unverändert  
Z - Unverändert  
N - Unverändert

---

BSET	Bit testen und setzen	BSET
------	-----------------------	------

---

Notation:

1.)	BSET	Dn,<ea>
2.)	BSET	#<datum>,<ea>

Objekt-Größe: Byte für Objekte im Speicher,  
Long für Objekte in Daten-Register.

Funktion: s.u.

BCLR testet ein bit von Operand-2 und setzt entsprechend dem Ergebnis das Z-bit auf 0 oder 1. Das gewählte bit im Operand-2 wird gesetzt. Ist Operand-2 ein Daten-Register, so erfolgt die Angabe der bit-Nummer modulo-32, andernfalls modulo-8.

### Flags:

X - Unverändert.  
C - Unverändert.  
V - Unverändert.  
Z - Gesetzt, falls das getestete bit = 0, sonst rückgesetzt.  
N - Unverändert.

#### 1.) Adressierungsarten für Operand-2: <ee>

Dn	(An)+	d(An,Ri)	.
.	~(An)	Abs.W	.
(An)	d(An)	Abs.L	.

#### 2.) Adressierungsarten für Operand-2: <ea>

Dn	(An)+	d(An,Ri)	.
.	~(An)	Abs.W	.
(An)	d(An)	Abs.L	.

---

BSR	Verzweige in Unterprogramm	BSR
-----	----------------------------	-----

---

Notation:        BSR        <Sprungziel>

Objekt-Größe:    Byte, Word

Funktion:        PC ---> (SP)  
                   PC + Sprungziel-Distanz ---, PC

BSR führt einen unbedingten Sprung zu einem Unterprogramm aus. Die Angabe des Sprungziels erfolgt relativ zum aktuellen PC-Stand (Distanz). Die Rücksprung-Adresse wird auf den System-Stack gelegt.

Zulässige Werte für <Sprungziel-Distanz> sind:

- Byte: vorzeichenbehaftete ganze Zahl (8-bit),  
       ( -128 ... -1, 1 ... +127 )  
       beachte: 0 ist nicht enthalten !
- Word: vorzeichenbehaftete ganze Zahl (16-bit),  
       ( -32768 ... 0 ... +32767 )

Unterprogramm-Aufrufe auf die unmittelbar folgende Adresse sind nur mit einer Distanz-Angabe in der Größe 'Word' möglich.

#### Flags:

- X - Unverändert
- C - Unverändert
- V - Unverändert
- Z - Unverändert
- N - Unverändert

BTST	Bit-Test	BTST
------	----------	------

Notation:

- 1.) BTST Dn,<aa>  
 2.) BTST \*(datum),<aa>

Objekt-Größe:

Byte für Objekte im Speicher,  
 Long für Objekte in Daten-Registern.

Funktion:

(Testet bit nn des Ziel-Objektes) ---> Z

BTST testet das ausgewählte bit des Ziel-Objektes und gibt das Ergebnis im Z-Flag wieder. Operand-1 gibt die bit-Nummer an, Operand-2 das Ziel-Objekt. Sofern Operand-2 ein Datenregister ist, erfolgt die Angabe der bit-Position modulo-32, sonst modulo-8. Bei Datenregistern können also 32 bits getestet werden, in Speicherplatz-Variablen nur 8 bit.

Flags:

- X - Unverändert.  
 C - Unverändert.  
 V - Unverändert.  
 Z - Wird mit dem inversen Inhalt des getesteten bits geladen.  
 N - Unverändert.

## 1.) Adressierungsarten für Operand-2:

Dn	(An)+	d(An,Ri)	d(\$)
.	-(An)	Abs.W	d(\$,Ri)
(An)	d(An)	Abs.L	.

## 2.) Adressierungsarten für Operand-2:

Dn	(An)+	d(An,Ri)	d(\$)
.	-(An)	Abs.W	d(\$,Ri)
(An)	d(An)	Abs.L	.



CHK

Teste Register auf Wertebereich

CHK

Notation: CHK <ea>,DnObjekt-Größe: WordFunktion: Teste, ob Dn < 0 oder > (<ea>)  
falls ja: ---> TRAP (CHK-Vektor)

Es wird getestet, ob der Inhalt des Daten-Registers im Bereich 0 ... (<ea>) liegt. Ist das nicht der Fall, wird ein TRAP generiert.

Flags:

- X - Unverändert.
- C - Nicht definiert.
- V - Nicht definiert.
- Z - Nicht definiert.
- N - Gesetzt, falls Dn < 0, rückgesetzt, falls Dn > (<ea>), sonst nicht definiert.

Adressierungsarten für Operand-1: &lt;ea&gt;

Dn	(An)+	d(An,Ri)	d(\$)
.	-(An)	Abs.H	d(\$,Ri)
(An)	d(An)	Abs.L	Imm

---

CLR	Setze Operand zurück	CLR
-----	----------------------	-----

---

Notation: CLR <ea>

Objekt-Größen: Byte, Word, Long

Funktion: 0 ---> Ziel-Operand

Setzt alle bits des Ziel-Operanden auf 0.

#### Flags:

X - Unverändert.  
 C - Rückgesetzt.  
 V - Rückgesetzt.  
 Z - Gesetzt.  
 N - Rückgesetzt.

Adressierungsarten für Ziel-Operand: <ea>

Dn	(An)+	d(An,Ri)	.
.	-(An)	Abs.W	.
(An)	d(An)	Abs.L	.

---

CMP	Vergleiche	CMP
-----	------------	-----

---

Notation: CMP (ea),Dn

Objekt-Größe: Byte, Word, Long

Funktion: (op1) - (op2) ---> Flags

Vergleicht Operand-1 und Operand-2. Als Ergebnis des Vergleichs werden die Condition-Codes gesetzt.

#### Flags:

- X - Unverändert.
- C - Gesetzt wenn Borrower erzeugt wird, sonst rückgesetzt.
- V - Gesetzt wenn Überlauf erzeugt wird, sonst rückgesetzt.
- Z - Gesetzt bei Gleichheit der Operanden, sonst rückgesetzt.
- N - Gesetzt wenn Vergleich zu negativem Ergebnis führt, sonst rückgesetzt.

Adressierungsarten für Operand-1: (ea)

Dn	(An)+	d(An,R1)	d(%)
An	-(An)	Abs.W	d(% ,R1)
(An)	d(An)	Abs.L	Imm

Objekt-Größen für Adress-Register direkt: Word und Long

CMPA	Vergleiche Adresse	CMPA
------	--------------------	------

Notation: CMPA <ea>,An

Objekt-Größe: Word, Long

Funktion: (op1) - (op2) ---> Flags

Vergleiche Operand-1 und Operand-2. Als Ergebnis des Vergleichs werden die Condition-Codes gesetzt. Operanden der Größe 'Word' werden unter Beibehaltung des Vorzeichens auf 32-bit erweitert:

z.B: (positiv)

```
0100 1000 1110 0011 --> 0000 0000 0000 0000 0100 1000 1110 0011
 4      8      E      3 --> 0 0 0 0 4 8 E 3
```

z.B: (negativ)

```
1001 1000 1110 0011 --> 1111 1111 1111 1111 1001 1000 1110 0011
 9      8      E      3 --> F F F F 9 8 E 3
```

### Flags:

- X - Unverändert.
- C - Gesetz wenn Borrow erzeugt wird, sonst rückgesetzt.
- V - Gesetz wenn Überlauf erzeugt wird, sonst rückgesetzt.
- Z - Gesetz bei Gleichheit der Operanden, sonst rückgesetzt.
- N - Gesetz wenn Vergleich zu negativem Ergebnis führt, sonst rückgesetzt.

Adressierungsarten für Operand-1: <ea>

Dn	(An)+	d(An,Ri)	d(\$)
An	-(An)	Abs.W	d(\$,Ri)
(An)	d(An)	Abs.L	Imm

CMPI	Vergleiche mit Konstante	CMPI
------	--------------------------	------

Notation: CMPI  $\{ccdstarm\}, \{ea\}$

Objekt-Größe: Byte, Word, Long

Funktion: (op1) - (op2) > Flags

Vergleicht Operand 1 und Operand 2. Als Ergebnis des Vergleichs werden die Condition-Codes gesetzt.

#### Flags:

- X - Unverändert.
- C - Gesetzt wenn Borrow erzeugt wird, sonst rückgesetzt.
- V - Gesetzt wenn Überlauf erzeugt wird, sonst rückgesetzt.
- Z - Gesetzt bei Gleichheit der Operanden, sonst rückgesetzt.
- N - Gesetzt wenn Vergleich zu negativem Ergebnis führt, sonst rückgesetzt.

Adressierungsarten für Operand-2: <ea>

[n]	(An)+	d(An,R1)	,
,	-(An)	abs.W	,
(An)	d(An)	Abs.L	,

---

CMPM	Vergleiche Speicherplätze	CMPM
------	---------------------------	------

---

Notation: CMPM (Ax)+,(Ay)+

Objekt-Größe: Byte, Word, Long

Funktion: (op1) - (op2) ----> Flags

CMPM vergleicht zwei Speicherplätze miteinander, das Ergebnis wird in den Flags wiedergegeben. Speicherplätze werden nicht verändert.

#### Flags:

- X - Unverändert.
- C - Gesetzt wenn Borrower erzeugt wird, sonst rückgesetzt.
- V - Gesetzt wenn Überlauf erzeugt wird, sonst rückgesetzt.
- Z - Gesetzt bei Gleichheit der Operanden, sonst rückgesetzt.
- N - Gesetzt wenn Vergleich zu negativem Ergebnis führt, sonst rückgesetzt.

---

**DBcc      Teste, decrementiere und verzweige      DBcc**

---

Notation:                      DBcc      [n, <Sprungziel>

Objekt-Größe:      Word

Funktion:

```

'cc' erfüllt: --> PC + 2 => PC
'cc' nicht erfüllt:
    Dn-1 --> Dn
    Falls Dn = -1:
        --> PC + 2 => PC
    Falls Dn < -1:
        --> PC + <Sprungdistanz> => PC

```

DBcc ist ein Schleifen-Kontroll Befehl. Er führt den Test einer Bedingung durch, decrementiert ein Datenregister und verzweigt im Programm. Eine Programmschleife kann daher sowohl von einer erfüllten Abbruch-Bedingung 'cc' beendet werden, als auch durch Erreichen eines bestimmten Zählerstandes im angegebenen D-Register.

Ist die Bedingung 'cc' erfüllt, so wird das Programm beim nächsten Befehl (PC + 2 => PC) fortgesetzt. Ist die Bedingung nicht erfüllt, so wird das Datenregister (operand-1) decrementiert und auf -1 getestet. Bei Erreichen dieses Zählerstandes ist die Programmschleife beendet, die Programmausführung wird beim nächsten Befehl fortgesetzt. Solange der Zählerstand -1 in Dn noch nicht erreicht ist, erfolgt die Verzweigung zum angegebenen Sprungziel.

Mögliche Bedingungen 'cc':

Kurzt:	Bedeutung:	Code:	logische Gleichung:
T	- True	0000	1
F	- False	0001	0
HI	- High	0010	$C * \bar{Z}$
LS	- Low or same	0011	$\bar{C} + Z$
CC	- Carry clear	0100	C
CS	- Carry set	0101	$\bar{C}$
NE	- Not equal	0110	$\bar{Z}$
EQ	- Equal	0111	Z
VC	- Overflow clear	1000	V
VS	- Overflow set	1001	$\bar{V}$
PL	- Plus	1010	N
MI	- Minus	1011	$\bar{N}$
GE	- Greater or equal	1100	$N * \bar{V} + \bar{N} * \bar{V}$
LT	- Less than	1101	$N * \bar{V} + \bar{N} * \bar{V}$
GT	- Greater than	1110	$N * \bar{V} * \bar{Z} + \bar{N} * \bar{V} * \bar{Z}$
LE	- Less or equal	1111	$Z + N * \bar{V} + \bar{N} * \bar{V}$

Flags:

-- Unverändert --

---

DIVS	Division mit Vorzeichen	DIVS
------	-------------------------	------

---

Notations: DIVS (ea),Dn

Objekt-Größe: Word

Funktion: (op2) / (op1) ---> (op2)

Dividiert ein Long-Word durch ein Word. Das Ergebnis ist von der Größe 'Word', das niederwertige Word enthält den Quotienten, das höherwertige den Rest:

bit-Nr:	31	16	15	0
	-----		-----	
	:	Rest	:	Quotient
	-----		-----	

Der Quotient liegt vorzeichenrichtig vor, der Rest trägt das Vorzeichen des Dividenden. (Ausnahme: Rest = 0)  
Division durch 0 führt zur IAPAP-Ausführung, bei Überlauf bleiben die Operanden unverändert.

#### Flags:

- X - Unverändert.
- C - Rückgesetzt.
- V - Gesetzt falls Überlauf, sonst rückgesetzt.
- Z - Gesetzt falls Quotient = 0, sonst rückgesetzt.  
Bei Überlauf: undefiniert.
- N - Gesetzt falls Quotient negativ, sonst rückgesetzt.  
Bei Überlauf: undefiniert.

Adressierungsarten für Operand-1: (ea)

Dn	(An)+	d(An,R1)	d(\$)
.	-(An)	Abs.W	d(\$,R1)
(An)	d(An)	Abs.L	Imm



DIUU	Division ohne Vorzeichen	DIUU
------	--------------------------	------

Notation: DIUU <ea>,On

Objekt-Größe: Word

Funktion: (op2) / (op1) ---> (op2)

Dividiert ein Long-Word durch ein Word. Das Ergebnis ist von der Größe 'Word', das niederwertige Wort enthält den Quotienten, das höherwertige den Rest:

bit-Nr:	31	16	15	0
	-----		-----	
	:	Rest	:	Quotient
	-----		-----	

Division durch 0 führt zur TRAP-Ausführung, bei Überlauf bleiben die Operanden unverändert.

#### Flags:

- X - Unverändert.
- C - Rückgesetzt.
- V - Gesetzt falls Überlauf, sonst rückgesetzt.
- Z - Gesetzt falls Quotient = 0, sonst rückgesetzt.  
Bei Überlauf: undefiniert.
- N - Gesetzt falls Quotient negativ, sonst rückgesetzt.  
Bei Überlauf: undefiniert.

Adressierungsarten für Operand-1: <ea>

Dn	(An)+	d(An,R1)	d(\$)
.	-(An)	Abs.W	d(\$,R1)
(An)	d(An)	Abs.L	Imm

---

EOR	Exklusiv-ODER Funktion	EOR
-----	------------------------	-----

---

Notations: EOR Dn,<ea>

Objekt-Größe: Byte, Word, Long

Funktion: (op1) EOR (op2) ---> (op2)

EOR führt bitweise die Exklusiv-Oder Funktion aus.

Beispiel: (WORD)

bit-15	0	
	0110 1011 1011 0110	6BE6
(EOR)	1001 0100 1011 1100	94BC
	<hr/> 1111 1111 0000 1010	<hr/> FF0A

### Flags:

- X - Unverändert.
- C - Rückgesetzt.
- V - Rückgesetzt.
- Z - Gesetzt falls Ergebnis = 0, sonst rückgesetzt.
- N - Wird mit dem höchstwertigen bit des Ergebnisses geladen.

Adressierungsarten für Operand-2: <ea>

Dn	(An)+	d(An,Ri)	.
.	-(An)	Abs.W	.
(An)	d(An)	Abs.L	.

EORl	Exklusiv-Oder mit Konstante	EORl
------	-----------------------------	------

Notation: EORl #(<datum>,<ea>

Objekt-Größe: Byte, Word, Long

Funktion: (op1) EORl (op2) ---> (op2)

EORl führt bitweise die Exklusiv-Oder Funktion mit einer Konstanten aus.

Beispiel: (WORD)

bit-15	0
0110 1011 1011 0110	68B6
(EOR) 1001 0100 1011 1100	94BC
<hr/> 1111 1111 0000 1010	<hr/> FF0A

#### Flags:

- X - Unverändert.
- C - Rückgesetzt.
- V - Rückgesetzt.
- Z - Gesetzt falls Ergebnis = 0, sonst rückgesetzt.
- N - Wird mit dem höchstwertigen bit des Ergebnisses geladen.

Adressierungsarten für Operand-2: (ea)

Dn	(An)+	d(An,Pl)	.
.	-(An)	Abs.H	.
(An)	d(An)	Abs.L	.

---

<b>EORI</b> <b>to CCR</b>	<b>Exklusiv-ODER mit Konst. u. CCR</b>	<b>EORI</b> <b>to CCP</b>
------------------------------	--	------------------------------

---

Notation: EORI #<datum>,CCR

Objekt-Größe: Byte

Funktion: (op1) EOR CCR ---> CCR

EORI führt bitweise die Exklusiv-Oder Funktion zwischen einer Konstanten und dem CCR aus.

Beispiel: (Byte)

bit=7	0	
	1010 1101	AD
(EOR)	1001 0100	94
	<hr/> 0011 1001	<hr/> 39

#### Flags:

- X - Invertiert falls bit 4 der Konstanten = 1, sonst unverändert.
- C - Invertiert falls bit 4 der Konstanten = 1, sonst unverändert.
- V - Invertiert falls bit 4 der Konstanten = 1, sonst unverändert.
- Z - Invertiert falls bit 4 der Konstanten = 1, sonst unverändert.
- N - Invertiert falls bit 4 der Konstanten = 1, sonst unverändert.

EORI to SR	Exklusiv-ODER mit konst. u. SR - privilegierter Befehl -	EORI to SR
---------------	---	---------------

Notation: EORI #<datum>,SR

Objekt-Größe: Word

Funktion: Falls Supervisor-Modus  
(npl) EOR SR ---> SR  
andernfalls IRRP ausführen.

EORI führt bitweise die Exklusiv-Oder Funktion zwischen einer Konstanten und dem SR aus.

Beispiel: (Word)

bit-15	0	
	1010 0101 0000 1111	A50F
(EOR)	1111 0000 1011 1100	F0BC
	<hr/> 0101 0101 1011 0000	<hr/> 55B0

#### Flags:

- X - Invertiert falls bit 4 der Konstanten = 1, sonst unverändert.
- C - Invertiert falls bit 4 der Konstanten = 1, sonst unverändert.
- V - Invertiert falls bit 4 der Konstanten = 1, sonst unverändert.
- Z - Invertiert falls bit 4 der Konstanten = 1, sonst unverändert.
- N - Invertiert falls bit 4 der Konstanten = 1, sonst unverändert.

---

EXG

---

Vertausche Register

EXG

---

Notation: EXG Px,RyObjekt-Größe: LongFunktion: (op1) (---) (op2)

EXG vertauscht die Inhalte zweier Register.

Flags:

- Unverändert -



---

 JMP

Sprung

JMP

---

Notation: JMP <ea>Objekt-Größe: -Funktion: <ea> ---> PC

JMP führt einen unbedingten Sprung zu, der in <ea> angegebenen Adresse aus.

Flags:

- Unverändert -

Adressierungsarten für &lt;ea&gt;:

.	.	d(An, Ri)	d(\$)
.	.	Abs.H	d(\$, Ri)
(An)	d(An)	Abs.L	.



---

JSR	Sprung in Unterprogramm	JSR
-----	-------------------------	-----

---

Notation: JSR <ea>

Objekt-Größe: -

Funktion: PC ---> (SP)  
<ea> ---> PC

Führt einen unbedingten Sprung in ein Unterprogramm aus. Die Rücksprung-Adresse wird auf den Stack geschrieben.

Flags:

- Unverändert -

Adressierungsarten für <ea>:

.	.	d(An,Ri)	d(0)
.	.	Abs.W	d(0,Ri)
(An)	d(An)	Abs.L	.

---

LEA	Lade effektive Adresse	LEA
-----	------------------------	-----

---

Notations: LEA <ee>,An

Objekt-Größe: Long

Funktion: ea --> An

Lädt eine effektive Adresse ee in das spezifizierte Adress-Register.

### Flags:

- Unverändert -

Adressierungsorten für <ea>:

.	.	d(An,Ri)	d(%)
.	.	Abs.W	d(% , Ri)
(An)	d(An)	Abs.L	.

---

LINK

Rette SP, lege neuen Stack an

LINK

---

Notation: LINK An, \* <Adress-Distanz>Objekt-Größe: -

Funktion:

An --> -(SP)  
SP --> An  
SP + <Adress-Distanz> --> SP

LINK führt nacheinander folgende Operationen durch:

- 1.) Der Inhalt des Adress-Registers An wird auf dem akt. Stack abgelegt.
- 2.) Der Stack-Pointer wird in das Adress-Register An gerettet.
- 3.) Stack-Pointer und <Adress-Distanz> werden addiert, das Ergebnis steht im Stack-Pointer.

Flags:

- Unverändert -

LSL	Logisch links Schieben	LSL
-----	------------------------	-----

Notations:

- 1.) LSL      Dx,Dy
- 2.) LSL      #<datum>,Dy
- 3.) LSL      <ee>

Objekt-Größe:

- 1.) Byte, Word, Long
- 2.) Byte, Word, Long
- 3.) Word

Funktion:

Ziel-Objekt um nn-Stellen links schieben

Das Ziel-Objekt (Operand-2 in den Fällen 1. und 2., Operand-1 im Fall 3.) wird um eine Anzahl (nn) bits logisch nach links geschoben. Die Anzahl (nn) der Schiebeoperation ergibt sich zu:

- 1.) Anzahl steht im Register Dx, (modulo 64)
- 2.) Anzahl wird durch den unmittelbaren Operanden angegeben, mögliche Werte: 1..8
- 3.) Anzahl = 1

```

-----
: C : <---
-----
      !
      !<----- Ziel - Objekt : <-----! 0 :
      !
      !
-----
: X : <---
-----

```

Flags:

- X - Wird mit dem zuletzt herausgeschobenen bit geladen.
- C - Wird mit dem zuletzt herausgeschobenen bit geladen, wird rückgesetzt, falls Anzahl der Schiebeoperationen = 0.
- V - Gesetzt falls das höchstwertige bit während der Ausführung der Schiebeoperation mindestens einmal verändert wurde.
- Z - Gesetzt falls das Ergebnis = 0, sonst rückgesetzt.
- N - Wird mit dem höchstwertigen bit des Ergebnisses geladen.

Adressierungsorten für Operand-1: <ee>

.	(An)+	d(An,Ri)	.
.	-(An)	Abs.W	.
(An)	d(An)	Abs.L	.



---

MOVE	übertrage Daten	MOVE
------	-----------------	------

---

Notation: MOVE (ea), (ea)

Objekt-Größe: Byte, Word, Long

Funktion: (op1) --> (op2)

überträgt das von OP-1 adressierte Datum in den von OP-2 angegebenen Ort. Als Quelle kann ein Datenregister oder eine Speicherstelle angegeben werden, Zielort kann ein D- oder A-Register oder eine Speicherstelle sein.

#### Flags:

- X - Unverändert.
- C - Rückgesetzt.
- V - Rückgesetzt.
- Z - Gesetzt, falls Objekt = 0, sonst rückgesetzt.
- N - Gesetzt, falls Objekt < 0, sonst rückgesetzt.

Adressierungsarten für Operand-1: (ea)

Dn	(An)+	d(An, Ri)	.
.	-(An)	Abs.W	.
(An)	d(An)	Abs.L	.

Adressierungsarten für Operand-2: (ea)

Dn	(An)+	d(An, Ri)	d(\$)
An	-(An)	Abs.W	d(\$, Ri)
(An)	d(An)	Abs.L	Imm

Objekt-Größen für Adress-Register direkt: Word und Long

---

 MOVE  
to CCR

übertrage Daten nach CCR

MOVE  
to CCR

---

Notation: MOVE <ea>,CCRObjekt-Größe: WordFunktion: (op1) ---> CCR

Überträgt das von OP-1 adressierte Datum in das CCR (Flag-Register). Als Quelle kann ein Datenregister oder eine Speicherstelle angegeben werden.

Flags:

- X - Wird mit bit 4 des übertragenen Objekts geladen.
- C - Wird mit bit 0 des übertragenen Objekts geladen.
- V - Wird mit bit 1 des übertragenen Objekts geladen.
- Z - Wird mit bit 2 des übertragenen Objekts geladen.
- N - Wird mit bit 3 des übertragenen Objekts geladen.

Adressierungsarten für Operand-1: &lt;ea&gt;

En	(An)+	d(An,R1)	d(PC)
.	-(An)	Abs.W	d(PC,R1)
(An)	d(An)	Abs.L	Imm

---

MOVE  
to SR

übertrage Daten nach SR  
- privilegierter Befehl -

MOVE  
to SR

---

Notation: MOVE <ea>,SR

Objekt-Größe: Word

Funktion: Falls Supervisor-Modus  
(op1) ---> SR  
andernfalls TRAP ausführen.

Überträgt das von OP-1 adressierte Datum in das SR (Status-Register). Als Quelle kann ein Datenregister oder eine Speicherstelle angegeben werden.

### Flags:

- X - Wird mit bit 4 des übertragenen Objekts geladen.
- C - Wird mit bit 0 des übertragenen Objekts geladen.
- V - Wird mit bit 1 des übertragenen Objekts geladen.
- Z - Wird mit bit 2 des übertragenen Objekts geladen.
- N - Wird mit bit 3 des übertragenen Objekts geladen.

Adressierungsarten für Operand-1: <ea>

Dn	(An)+	d(An,Ri)	d(PC)
.	-(An)	Abs.W	d(PC,Ri)
(An)	d(An)	Abs.L	Imm



---

MOVE	übertrage Daten von SR	MOVE
from SR		from SR

---

Notation: MOVE SR, (ea)

Objekt-Größe: Word

Funktion: SR ---> (op2)

Überträgt den Inhalt des Status-Registers (SR) an den von OP-2 festgelegten Ort. Zielort kann ein sowohl Datenregister als auch eine Speicherstelle sein.

#### Flags:

- Unverändert -

Adressierungsarten für Operand-2: (ea)

Dn	(An)+	d(An, Ri)	.
.	-(An)	Abs.W	.
(An)	d(An)	Abs.L	.

---

MOVE  
USP

Übertrage von/nach USP  
- privilegierter Befehl -

MOVE  
USP

---

Notation:      MOVE      USP,An  
                 MOVE      An,USP

Objekt-Größe:   Long

Funktion:        Falls Supervisor-State:  
                      USP ---> An  
                      An    ---> USP  
                      andernfalls TRAP ausführen.

überträgt den Inhalt des User-Stackpointers (USP) von bzw. in das  
angegebene Adress-Register.

Flags:

- Unverändert -

MOVEA	übertrage Adresse	MOVEA
-------	-------------------	-------

Notation:      MOVEA      (ea),An

Objekt-Größe:    Word, Long

Funktion:        (op1) ← op2    An

überträgt das von OP1 adressierte Datum in das angegebene A-Register. Als Quelle kann ein Register, eine Speicherstelle oder eine Konstante angegeben werden.

#### Flags:

- Unverändert -

Adressierungsarten für Operand-1: (ea):

On	(An)+	d(An,R1)	d(PC)
An	-(An)	Abs.W	d(PC,R1)
(An)	d(An)	Abs.L	Imm

---

MOVEM	Übertrage mehrere Register	MOVEM
-------	----------------------------	-------

---

Notation:      MOVEM      Register-Liste,<ea>  
                  MOVEM      <ea>,Register-Liste

Objekt-Größe:    Word, Long

Funktion:        s.u.

Die ausgewählten Register werden nacheinander in den Speicher, beginnend ab der angegebenen Adresse, übertragen. Objekt-Größe kann Word oder Long sein, entsprechend werden 16 oder 32 bit übertragen.

Bei der Übertragung vom Speicher in die Register werden Objekte von Word Größe vorzeichenrichtig auf 32-bit erweitert.

### Flags:

- Unverändert -

Adressierungsarten für Operand-1: <ea>

.	(An)+	d(An,Ri)	d(\$)
.	.	Abs.W	d(\$,Ri)
(An)	d(An)	Abs.L	.

Adressierungsarten für Operand-2: <ea>

.	.	d(An,Ri)	.
.	-(An)	Abs.W	.
(An)	d(An)	Abs.L	.



---

MOVEQ

Übertrage schnell

MOVEQ

---

Notations:      MOVEQ      #<datum>,DnObjekt-Größe:    LongFunktion:        #<datum>    ---> Dn

Lädt das angegebende Datenregister mit einer 8-bit Konstanten.

Flags:

- X - Unverändert.
- C - Stets rückgesetzt.
- V - Stets rückgesetzt.
- Z - Besetzt, falls Objekt = 0, sonst rückgesetzt.
- N - Wird mit bit 7 der Konstanten geladen.

---

MULS	Multiplikation mit Vorzeichen	MULS
------	-------------------------------	------

---

Notation: MULS <ea>,Dn

Objekt-Größe: Word

Funktion: (op1) \* (op2) ---> (op2)

Multipliziert 2 vorzeichenbehaftete Objekte der Größe Word. Das höherwertige Word des Registers wird ignoriert. Dn enthält nach der Operation das vorzeichenrichtige 32-bit lange Produkt.

#### Flags:

- X - Unverändert.
- C - Rückgesetzt.
- V - Rückgesetzt.
- Z - Gesetzt, falls das Ergebnis = 0, sonst rückgesetzt.
- N - Wird mit bit-31 des Ergebnisses geladen.

Adressierungsarten für Operand-1: <ea>

Dn	(An)+	d(An,R1)	d(\$)
.	-(An)	Abs.W	d(\$,R1)
(An)	d(An)	Abs.L	Imm

**MULU**

Multiplikation ohne Vorzeichen

**MULU**Notation: MULU <ea>,DnObjekt-Größe: WordFunktion: (op1) \* (op2) ---> (op2)

Multipliziert 2 vorzeichenlose Objekte der Größe Word. Das höherwertige Word des Registers wird ignoriert. Dn enthält nach der Operation das vorzeichenlose 32-bit lange Produkt.

Flags:

- X - Unverändert.
- C - Rückgesetzt.
- V - Rückgesetzt.
- Z - Gesetzt, falls das Ergebnis = 0, sonst rückgesetzt.
- N - Wird mit bit-31 des Ergebnisses geladen.

Adressierungsarten für Operand-1: &lt;ea&gt;

Dn	(An)+	d(An,R1)	d(£)
.	-(An)	Abs.W	d(£,R1)
(An)	d(An)	Abs.L	Inm



NBCD	Dezimal Negation	NBCD
------	------------------	------

Notation: NBCD (ea)

Objekt-Größe: Byte

Funktion: 0 - (opl) - X ---> (opl)

Negiert eine Dezimalzahl im BCD-Code. X-bit = 0 führt zur Bildung des 10-er Komplements, X-bit = 1 führt zur Bildung des 9-er Komplements.

Beispiel: ( 1 Byte = 2 Dezimal-Stellen )

		BCD:		dez:
Zahl:	=	0001 1000	=	18
1-er Komplement:		1110 0111		
+AA (hex):	=	1010 1010		
9-er Komplement:	=	1000 0001	=	81
+1:	=	0000 0001		
10-er Komplement:	=	1000 0010	=	82

### Flags:

- X - Gesetzt wie Carry-bit.
- C - Gesetzt, falls ein Dezimal-Borger erzeugt wird, sonst rückgesetzt.
- V - undefiniert.
- Z - Rückgesetzt, falls Ergebnis (<) 0, sonst unverändert.
- N - undefiniert.

Adressierungsarten für Operand-1: (ea)

Dn	(An)+	d(An,R1)	.
.	-(An)	abs.W	.
(An)	d(An)	abs.L	.

---

NEG	Negation	NEG
-----	----------	-----

---

Notation:            NEG    (<ea>)

Objekt-Größe:    Byte, Word, Long

Funktion:            0 - (<op1> ---) (<op1>)

Negiert den von (<ea>) adressierten Operanden mit der angegebenen Größe Byte, Word oder Long.

Beispiel:            (Byte)

		binär:		hex:
Zahl:	=	0001 1000	=	18
1-er Komplement:		1110 0111	=	E7
+1:		0000 0001	=	01
2-er Komplement:		<u>1110 1000</u>	=	<u>E8</u>

### Flags:

- X - Wird gesetzt wie das Carry-bit.
- C - Rückgesetzt, falls Ergebnis = 0, sonst gesetzt.
- V - Gesetzt, falls ein Überlauf erzeugt wird, sonst rückgesetzt.
- Z - Gesetzt, falls Ergebnis = 0, sonst rückgesetzt.
- N - Gesetzt, falls das Ergebnis < 0, sonst rückgesetzt.

Adressierungsarten für Operand-1: (<ea>)

Dn	(An)+	d(An,Ri)	.
.	-(An)	Abs.W	.
(An)	d(An)	Abs.L	.

NEGX	Negation mit 1-Flag	NEGQ
------	---------------------	------

Notation:      NEGX      (ea)

Objekt-Größe:    Byte, Word, Long

Funktion:        0 - (op1) → X    ---> (op1)

Negiert den von 'ea' adressierten Operanden mit der angegebenen Größe Byte, Word oder Long. Je nach Inhalt des X-bits wird das 1-er Komplement (X=1) bzw. das 2-er Komplement (X=0) gebildet.

Beispiel:      (Byte)

		binär:		hex:
Zahl:	=	0001 1000	=	18
1-er Komplement:	=	1110 0111	=	E7
+1:		0000 0001	=	01
2-er Komplement:	=	<u>1110 1000</u>	=	<u>E8</u>

### Flags:

- X - Wird gesetzt wie das Carry-bit.
- C - Gesetzt, falls ein Borrow erzeugt wird, sonst zurückgesetzt.
- V - Gesetzt, falls ein Überlauf erzeugt wird, sonst zurückgesetzt.
- Z - Rückgesetzt, falls Ergebnis > 0, sonst unverändert.
- N - Gesetzt, falls das Ergebnis < 0, sonst zurückgesetzt.

Adressierungsarten für Operand 1: (ea)

En	(An)+	d(An, R1)	.
.	-(An)	abs.W	.
(An)	d(An)	abs.L	.

---

NOP	Keine Operation	NOP
-----	-----------------	-----

---

Notation: NOP

Objekt-Größe: -

Funktion: keine

Es wird keine Operation ausgeführt.

Flags:

- Unverändert -

INSTR	INSTR - Funktion	INSTR
<u>Notation:</u>	NOT	not
<u>Objekt-Größe:</u>	Byte, Word, Long	
<u>Funktion:</u>	NOT (op1) → ~ op1	

Invertiert den Operanden bitweise entsprechend der angegebenen Größe.

Beispiel: (Word)

		Binär:	hex:
Zahl:	=	0000 0000 0001 1000	= 0010
1er Komplement:	=	1111 1111 1110 0111	= FFEF

### Flags:

- X - Unverändert.
- C - Rückgesetzt.
- O - Rückgesetzt.
- Z - Gesetzt, falls das Ergebnis = 0, sonst rückgesetzt.
- N - Gesetzt, falls das Ergebnis > 0, sonst rückgesetzt.

Adressierungsarten für Operand-1: (ea)

ea	(An)+	d(ea,Rt)	.
.	-(An)	abs.W	.
(An)	d(An)	abs.L	.

OR	ODER - Funktion	OR
----	-----------------	----

Notation:      OR      <ea>,Dn  
                  OR      Dn,<ea>

Objekt-Größe:    Byte, Word, Long

Funktion:            (op1) ODER (op2)    --->    (op2)

Bildet die logische ODER-Funktion mit den beiden Operanden.

Beispiel:      (Word)

```

                1101 0010 0010 0011
ODER           1110 0011 1110 0000

```

---

```

                1111 0011 1110 0011

```

### Flags:

- X - Unverändert.
- C - Rückgesetzt.
- V - Rückgesetzt.
- Z - Gesetzt, falls Ergebnis = 0, sonst rückgesetzt.
- N - Gesetzt, falls das höchste Ergebnisbit gesetzt ist, sonst rückgesetzt.

Adressierungsarten für Operand-1: <ea>

Dn	(An)+	d(An,Ri)	d(†)
,	-(An)	Abs.W	d(†,Ri)
(An)	d(An)	Abs.L	Imm

Adressierungsarten für Operand-2: <ea>

.	(An)+	d(An,Ri)	.
.	-(An)	Abs.W	.
(An)	d(An)	Abs.L	.

---

ORI	ODER - Funktion mit Konstante	ORI
-----	-------------------------------	-----

---

Notation: ORI      \*(datum), (ea)

Objekt-Größe: Byte, Word, Long

Funktion: (op1) ODER (op2)    ---> (op2)

Bildet die logische ODER-Funktion mit den beiden Operanden.

Beispiel:      (Word)

	1101 0010 0010 0011
ODER	1110 0011 1110 0000
	<hr/> 1111 0011 1110 0011

### Flags:

- X - Unverändert.
- C - Rückgesetzt.
- U - Rückgesetzt.
- Z - Gesetzt, falls Ergebnis = 0, sonst rückgesetzt.
- N - Gesetzt, falls das höchste Ergebnisbit gesetzt ist, sonst rückgesetzt.

Adressierungsarten für Operand-2: (ea)

Dn	(An)+	d(An, Ri)	.
.	-(An)	Abs.W	.
(An)	d(An)	Abs.L	.

---

ORI to CCR	ODER - Funktion mit Flags	ORI to CCR
---------------	---------------------------	---------------

---

Notation: ORI      \*(<datum>),CCR

Objekt-Größe: Byte

Funktion:      \*(<datum>) ODER CCR    ---> CCR

Bildet die logische ODER-Funktion mit den beiden Operanden, das Ergebnis wird im niederen Byte des Status-Registers abgelegt.

Beispiel:      (Byte)

	0010 0011
ODER	1110 0000
	-----
	1110 0011

#### Flags:

- X - Gesetzt, falls bit-4 von \*(<datum>) = 1, sonst unverändert.
- C - Gesetzt, falls bit-0 von \*(<datum>) = 1, sonst unverändert.
- V - Gesetzt, falls bit-1 von \*(<datum>) = 1, sonst unverändert.
- Z - Gesetzt, falls bit-2 von \*(<datum>) = 1, sonst unverändert.
- N - Gesetzt, falls bit-3 von \*(<datum>) = 1, sonst unverändert.



---

ORI to SR	ODER - Funktion mit SR - privilegierter Befehl -	ORI to SR
--------------	---	--------------

---

Notation: ORI      #<datum>,SR

Objekt-Größe: Word

Funktion:      Falls Supervisor-Status:  
                 #<datum> ODER CCR    ---> CCR  
                 andernfalls TFRP ausführen.

Bildet die logische ODER-Funktion mit den beiden Operanden, das Ergebnis wird im Status-Register abgelegt.

Beispiel:      (Word)

	1101 0010 0010 0011
ODER	1110 0011 1110 0000
	<hr/> 1111 0011 1110 0011

### Flags:

- X - Gesetzt, falls bit-4 von #<datum> = 1, sonst unverändert.
- C - Gesetzt, falls bit-0 von #<datum> = 1, sonst unverändert.
- V - Gesetzt, falls bit-1 von #<datum> = 1, sonst unverändert.
- Z - Gesetzt, falls bit-2 von #<datum> = 1, sonst unverändert.
- N - Gesetzt, falls bit-3 von #<datum> = 1, sonst unverändert.

---

PEA	Bringe effektive Adr. auf Stack	PEA
-----	---------------------------------	-----

---

Notation: PEA <ea>

Objekt-Größe: Long

Funktion: (op1) ---> -(SP)

Berechnet die effektive Adresse und legt sie auf dem Stack ab.

### Flags:

- Unverändert -

Adressierungsarten für Operand-1: <ea>

.	.	d(An,R1)	d(\$)
.	.	Abs.W	d(\$,R1)
(An)	d(An)	Abs.L	.

---

RESET	Periphere Schaltungen rücksetzen - privilegierter Befehl -
-------	---

---

RESET

Notation: RESET

Objekt-Größe: -

Funktion: Falls Supervisor-Status:  
----> Signal auf RESET-Leitung  
andernfalls TRAP ausführen.

Periphere Schaltungen werden durch ein Signal auf der RESET-Leitung in den Grundzustand gebracht.

Flags:

- Unverändert -

ROL	Rotiere links	ROL
-----	---------------	-----

Notations:

- 1.) ROL Dx,Dy
- 2.) ROL \*(datum),Dy
- 3.) ROL <ea>

Objekt-Größe:

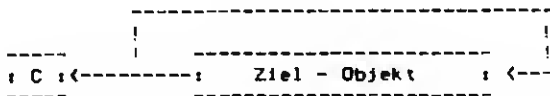
- 1.) Byte, Word, Long
- 2.) Byte, Word, Long
- 3.) Word

Funktion:

Ziel-Objekt um nn-Stellen nach links rotieren

Das Ziel-Objekt (Operand-2 in den Fällen 1. und 2., Operand-1 im Fall 3.) wird um eine Anzahl (nn) bits nach links rotiert. Die Anzahl (nn) der Rotieroperation ergibt sich zur

- 1.) Anzahl steht im Register Dx, (module 64)
- 2.) Anzahl wird durch den unmittelbaren Operanden angegeben, mögliche Werte: 1..8
- 3.) Anzahl = 1

Flags:

- X - Unverändert.
- C - Wird mit dem zuletzt herausgeschobenen bit geladen.
- V - Rückgesetzt.
- Z - Gesetzt falls das Ergebnis = 0, sonst rückgesetzt.
- N - Wird mit dem höchstwertigen bit des Ergebnisses geladen.

Adressierungsarten für Operand-1: <ea>

.	(An)+	d(An,R1)	.
.	-(An)	Abs.W	.
(An)	d(An)	Abs.L	.

ROR	Rotiere rechts	ROR
-----	----------------	-----

Notations:

- 1.) ROR      Dx,Dy
- 2.) ROR      \*(<datum>,Dy
- 3.) ROR      <ea>

Objekt-Größe:

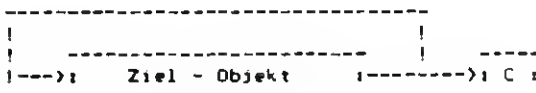
- 1.) Byte, Word, Long
- 2.) Byte, Word, Long
- 3.) Word

Funktion:

Ziel-Objekt um nn-Stellen nach rechts rotieren

Das Ziel-Objekt (Operand-2 in den Fällen 1. und 2., Operand-1 im Fall 3.) wird um eine Anzahl (nn) bits nach rechts rotiert. Die Anzahl (nn) der Rotieroperation ergibt sich zu:

- 1.) Anzahl steht im Register Dx, (modulo 64)
- 2.) Anzahl wird durch den unmittelbaren Operanden angegeben, mögliche Werte: 1..8
- 3.) Anzahl = 1

Flags:

- X - Unverändert.
- C - Wird mit dem zuletzt herausgeschobenen bit geladen.
- V - Rückgesetzt.
- Z - Gesetzt falls das Ergebnis = 0, sonst rückgesetzt.
- N - Wird mit dem höchstwertigen bit des Ergebnisses geladen.

Adressierungsarten für Operand-1: <ea>

.	(An)+	d(An,R1)	.
.	-(An)	Abs.W	.
(An)	d(An)	Abs.L	.

ROXL	Rotiere links mit X-Flag	ROXL
------	--------------------------	------

Notations:

- 1.) ROXL      Dx,Dy
- 2.) ROXL      #<datum>,Dy
- 3.) ROXL      <ea>

Objekt-Größe:

- 1.) Byte, Word, Long
- 2.) Byte, Word, Long
- 3.) Word

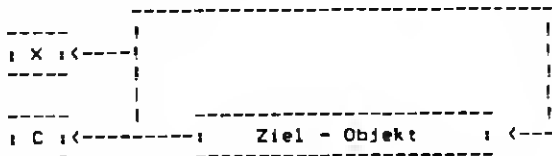
Funktion:

Ziel-Objekt um nn-Stellen nach links rotieren

Das Ziel-Objekt (Operand-2 in den Fällen 1. und 2., Operand-1 im Fall 3.) wird um eine Anzahl (nn) bits nach links rotiert. Die Anzahl (nn) der Rotieroperation ergibt sich zu:

- 1.) Anzahl steht im Register Dx, (modulo 64)
- 2.) Anzahl wird durch den unmittelbaren Operanden angegeben, mögliche Werte: 1..8
- 3.) Anzahl = 1

Sowohl das Carry-bit als auch das X-bit werden verändert:

Flags:

- X - Wird mit dem zuletzt herausgeschobenen bit geladen, unverändert falls keine Rotieroperation stattfand.
- C - Wird mit dem zuletzt herausgeschobenen bit geladen. Falls keine Rotieroperation stattfand wird es mit dem Wert des X-Flags geladen.
- V - Rückgesetzt.
- Z - Gesetzt falls das Ergebnis = 0, sonst rückgesetzt.
- N - Wird mit dem höchstwertigen bit des Ergebnisses geladen.

Adressierungsarten für Operand-1: <ea>

.	(An)+	d(An,R1)	.
.	-(An)	Abs.W	.
(An)	d(An)	Abs.L	.

ROXR	Rotiere rechts mit X-Flag	ROXR
------	---------------------------	------

Notation:

- 1.) ROXR Dx,Dy
- 2.) ROXR #<datum>,Dy
- 3.) ROXR <ea>

Objekt-Größe:

- 1.) Byte, Word, Long
- 2.) Byte, Word, Long
- 3.) Word

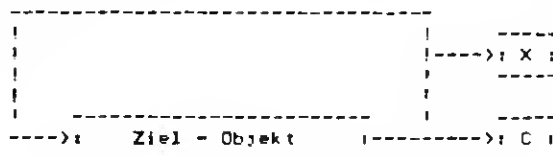
Funktion:

Ziel-Objekt um nn-Stellen nach rechts rotieren

Das Ziel-Objekt (Operand-2 in den Fällen 1. und 2., Operand-1 im Fall 3.) wird um eine Anzahl (nn) bits nach rechts rotiert. Die Anzahl (nn) der Rotieroperation ergibt sich zu:

- 1.) Anzahl steht im Register Dx, (modulo 64)
- 2.) Anzahl wird durch den unmittelbaren Operanden angegeben, mögliche Werte 1..8
- 3.) Anzahl = 1

Sowohl das Carry-bit als auch das X-bit werden verändert:

Flags:

- X - Wird mit dem zuletzt herausgeschobenen bit geladen, unverändert falls keine Rotieroperation stattfand.
- C - Wird mit dem zuletzt herausgeschobenen bit geladen. Falls keine Rotieroperation stattfand wird es mit dem Wert des X-Flags geladen.
- V - Rückgesetzt.
- Z - Gesetzt falls das Ergebnis = 0, sonst rückgesetzt.
- N - Wird mit dem höchstwertigen bit des Ergebnisses geladen.

Adressierungsarten für Operand-1: <ea>

.	(An)+	d(An,RI)	.
.	-(An)	Abs.W	.
(An)	d(An)	Abs.L	.

RTE

Rückkehr von Ausnahme  
- privilegierter Befehl -

RTE

Notation: RTEObjekt-Größe: -

Funktion: If Supervisor-Status:  
(SF)+ --> SR, (SP)+ --> PC  
andernfalls TRAP ausführen.

Das Status-Register und der Programm-Counter werden vom Stack geholt.

Flags:

- Werden mit den Werten geladen, die das oberste Word auf dem Stack enthielt -



---

RTR      Rückkehr mit Wiederherstellung der Flags    RTR

---

Notation:            RTR

Objekt-Größe:      -

Funktion:            (SP)+ --> CCR,    (SP)+ --> PC

Das Flag-Register und der Programm-Counter werden vom Stack geholt.

Flags:

- Werden mit den Werten geladen, die das oberste Word auf dem Stack enthielt -

---

RTS

Rückkehr von Unterprogramm

RTS

---

Notation: RTSObjekt-Größe: -Funktion: (SP)+ ---) PC

Der Programm-Counter wird mit dem obersten Wert auf dem Stack geladen.

Flags:

- Unverändert -

SBCD	Dezimal Subtraktion mit X-Flag	SBCD
------	--------------------------------	------

Notation:      SBCD      Dy,Dx  
                  SBCD      -(Ay),-(Ax)

Objekt-Größe:    Byte

Funktion:        (op2) - (op1) ~ X ---> (op2)

Subtrahiert Operand-1 und das X-Flag von Operand-2, das Ergebnis steht anschließend in Operand-2. Die Subtraktion erfolgt nach der BCD - Arithmetik, jedes Byte repräsentiert dabei 2 dezimale Ziffern.

Beispiele:

dezimal:	BCD:
63	0110 0011
- 27	- 0010 0111
<hr/> 36	<hr/> 0011 0110

#### Flags:

- X - Wird gesetzt wie das C-Flag.
- C - Gesetzt, falls ein dezimaler Borrow erzeugt wird, sonst rückgesetzt.
- V - Nicht definiert.
- Z - Rückgesetzt, falls das Ergebnis ( $\neq$  0), sonst unverändert.
- N - Nicht definiert.

Scc	Bedingtes Setzen eines Bytes	Scc
-----	------------------------------	-----

Notation: Scc <ea>

Objekt-Größe: Byte

Funktion: Falls Bedingung erfüllt:  
#0FFH ---) (op1)  
andernfalls:  
#0 ---) (op1)

Scc setzt das in <ea> spezifizierte Byte in Abhängigkeit von der Bedingung 'cc'. Ist 'cc' zutreffend, wird das Byte auf den Wert 0FFH (alle bits = 1), andernfalls auf 0 gesetzt.

Mögliche Bedingungen 'cc':

Kurzt:	Bedeutung:	Code:	logische Gleichung:
T	- True	0000	1
F	- False	0001	0
HI	- High	0010	$C * \bar{Z}$
LS	- Low or same	0011	$C + Z$
CC	- Carry clear	0100	$\bar{C}$
CS	- Carry set	0101	C
NE	- Not equal	0110	$\bar{Z}$
EQ	- Equal	0111	Z
VC	- Overflow clear	1000	$\bar{V}$
VS	- Overflow set	1001	V
PL	- Plus	1010	$\bar{N}$
MI	- Minus	1011	N
GE	- Greater or equal	1100	$N * \bar{V} + \bar{N} * \bar{V}$
LT	- less than	1101	$N * V + \bar{N} * V$
GT	- Greater than	1110	$N * V * \bar{Z} + \bar{N} * V * \bar{Z}$
LE	- Less or equal	1111	$Z + N * V + N * V$

### Flags:

- Unverändert -

Adressierungsarten für Operand-1: <ea>

Dn	(An)+	d(An,Ri)	.
.	-(An)	Abs.W	.
(An)	d(An)	Abs.L	.

---

**STOP**Lade Statusregister und halte  
- privilegierter Befehl -**STOP**

---

Notation:            STOP    #<datum>Objekt-Größe:    -Funktion:            Falls Supervisor-Status:  
                         #<datum> ---> SR, STOP  
                         andernfalls:  
                         TRAP

Die Konstante wird in das Status-Register geladen, der Programm-Zähler auf die nächste Instruktion gestellt und der Prozessor angehalten. Dieser Zustand wird beibehalten bis eine der drei folgenden Ausnahmen auftritt:

1. ein RESET-Signal trifft ein (---> RESET-Ausnahme),
2. es erfolgt eine Unterbrechungs-Anforderung mit höherer Priorität als den aktuellen Prozessorstatus (---> Interrupt-Ausnahme),
3. das Trace-bit ist gesetzt, es erfolgt beim Auftreten des STOP-Befehls die Bearbeitung der TRACE-Ausnahme.

Flags:

- Werden entsprechend #<datum> gesetzt -

SUB	Binär Subtraktion	SUB
-----	-------------------	-----

Notation: SUB    <ea>, Dn  
 SUB    Dn, <ea>

Objekt-Größe: Byte, Word, Long

Funktion: (op2) - (op1) ---> (op2)

Subtrahiert Operand-1 von Operand-2, Ergebnis steht anschließend in Operand-2.

#### Flags:

- X - Wird gesetzt wie das C-Flag.
- C - Gesetz, falls ein Borger erzeugt wird, sonst rückgesetzt.
- V - Gesetz, falls ein Überlauf erzeugt wird, sonst rückgesetzt.
- Z - Gesetz, falls das Ergebnis = 0, sonst rückgesetzt.
- N - Gesetz, falls das Ergebnis < 0, sonst rückgesetzt.

Adressierungsarten für Operand-1: <ea>

Dn	(An)+	d(An, Ri)	d(\$)
An	-(An)	Abs.W	d(\$, Ri)
(An)	d(An)	Abs.L	Imm

Objekt-Größen für Adress-Register direkt: nur Word und Long

Adressierungsarten für Operand-2: <ea>

.	(An)+	d(An, Ri)	.
.	-(An)	Abs.W	.
(An)	d(An)	Abs.L	.

---

SUBA	Adress Subtraktion	SUBA
------	--------------------	------

---

Notations: SUBA <ea>,An

Objekt-Größe: Word, Long

Funktion: (op2) - (op1) ----> (op2)

Subtrahiert Operand-1 von Operand-2 (= Adress-Register), das Ergebnis steht anschließend in Operand-2 (Adress-Register).

### Flags:

- Unverändert -

Adressierungsarten für Operand-1: <ea>

Dn	(An)+	d(An,R1)	d(\$)
An	-(An)	Abs.W	d(\$,R1)
(An)	d(An)	Abs.L	Imm

---

SUBI	Subtraktion mit Konstante	SUBI
------	---------------------------	------

---

Notation: SUBI #<datum>,<ea>

Objekt-Größe: Byte, Word, Long

Funktion: (op2) - (op1) ---> (op2)

Subtrahiert Operand-1 (= Konstante) von Operand-2, das Ergebnis steht anschließend in Operand-2.

#### Flags:

- X - Wird gesetzt wie das C-Flag.
- C - Gesetzt, falls ein Borrow erzeugt wird, sonst rückgesetzt.
- V - Gesetzt, falls ein Überlauf erzeugt wird, sonst rückgesetzt.
- Z - Gesetzt, falls das Ergebnis = 0, sonst rückgesetzt.
- N - Gesetzt, falls das Ergebnis < 0, sonst rückgesetzt.

Adressierungsarten für Operand-2: <ea>

Dn	(An)+	d(An,Ri)	.
,	-(An)	Abs.W	.
(An)	d(An)	Abs.L	.



---

SUBQ	Subtrahiere schnell	SUBQ
------	---------------------	------

---

Notation: SUBQ    #(<datum>), <ea>

Objekt-Größen: Byte, Word, Long

Funktion:    (op2) ← (op1)    --->    (op2)

Subtrahiert Operand-1 (= Konstante) von Operand-2, das Ergebnis steht anschließend in Operand-2. Der Wertebereich der Konstanten ist beschränkt auf: 1...8.

#### Flags:

- X - Wird gesetzt wie das C-Flag.
- C - Gesetzt, falls ein Borrow erzeugt wird, sonst rückgesetzt.
- V - Gesetzt, falls ein Überlauf erzeugt wird, sonst rückgesetzt.
- Z - Gesetzt, falls das Ergebnis = 0, sonst rückgesetzt.
- N - Gesetzt, falls das Ergebnis < 0, sonst rückgesetzt.

Adressierungsarten für Operand-2: <ea>

Dn	(An)+	d(An,R1)	.
An	-(An)	Abs.W	.
(An)	d(An)	Abs.L	.

Objekt-Größen für Adress-Register direkt:    nur Word und Long

---

**SUBX**

Subtraktion mit X-Flag

**SUBX**

---

Notations:

SUBX      Dy, Dx  
SUBX      -(Ay), -(Ax)

Objekt-Größe:

Byte, Word, Long

Funktion: $(op2) - (op1) - X \rightarrow (op2)$ 

Subtrahiert Operand-1 und das X-Flag von Operand-2, das Ergebnis steht anschließend in Operand-2.

Flags:

- X - Wird gesetzt wie das C-Flag.
- C - Gesetzt, falls ein Übertrag erzeugt wird, sonst rückgesetzt.
- V - Gesetzt, falls ein Überlauf erzeugt wird, sonst rückgesetzt.
- Z - Gesetzt, falls das Ergebnis = 0, sonst rückgesetzt.
- N - Gesetzt, falls das Ergebnis < 0, sonst rückgesetzt.

---

SWAP	Vertausche Registerhälften	SWAP
------	----------------------------	------

---

Notation: SWAP Dn

Objekt-Größe: Word

Funktion: low-Word <---> high-Word

Vertauscht die beiden Registerhälften im Data-Register n.

Flags:

- X - Unverändert.
- C - Rückgesetzt.
- V - Rückgesetzt.
- Z - Gesetzt, falls Ergebnis = 0, sonst rückgesetzt.
- N - Wird mit bit 31 geladen.

---

TAS	Teste und setze Operanden	TAS
-----	---------------------------	-----

---

Notation: TAS     <ea>

Objekt-Größe: Byte

Funktion: s.u.

Testet den in <ea> adressierten Operanden und setzt die Flags dementsprechend. Anschließend wird des bit-7 des Operanden auf 1 gesetzt.

#### Flags:

- X - Unverändert.
- C - Rückgesetzt.
- V - Rückgesetzt.
- Z - Gesetzt, falls der Operand = 0 war, sonst rückgesetzt.
- N - Wird mit dem ursprünglichen Wert von bits-7 des Operanden geladen.

Adressierungsarten für Operand-1: <ea>

Dn	(An)+	d(An,Ri)	.
.	-(An)	Abs.W	.
(An)	d(An)	Abs.L	.

---

TRAP	Trap	TRAP
------	------	------

---

Notations: TRAP #<vektor-nr>

Objekt-Größe: -

Funktion: PC --> -(SSP)  
SF --> -(SSP)  
(<vektor-nr>) --> PC

Trap löst die Exception-Bearbeitung (=Supervisor-Status) aus, Programm-Zähler und Status-Register werden auf den Stack geschrieben, der angegebene Trap-Vektor wird in den Programm-Zähler geladen. Zulässige Werte für <vektor-nr>: 0...15.

Flags:

- Unverändert -

---

UNLK

Hole Stackpointer zurück

UNLK

---

Notation:

UNLK    An

Objekt-Größe:

-

Funktion:An --> SP  
(SP)+ --> An

Der Stackpointer wird mit dem Inhalt des angegebenen Adress-Registers geladen, anschließend wird der oberste Wert vom Stack in das Adress-Register geschrieben.

Flags:

- Unverändert -

### 3. Muster-Programme

- beginnen auf der nächsten Seite -

---

TRAPV	Trap bei Überlauf	TRAPV
-------	-------------------	-------

---

Notation: TRAPV

Objekt-Größe: -

Funktion: Falls V-Flag = 1 --> TRAP

Falls das Overflow-Flag gesetzt ist wird ein TRAP ausgelöst, der TRAPV-Vektor wird geladen. Bei rückgesetzten V-Flag wird ein NOP ausgeführt.

Flags:

- Unverändert -



TST

Teste einen Operanden

TST

Notation: TST (ea)Objekt-Größe: Byte, Word, LongFunktion: Teste Operanden ---> Flags

Testet den Operanden und setzt die Flags entsprechend.

Flags:

- X - Unverändert.
- C - Rückgesetzt.
- V - Rückgesetzt.
- Z - Gesetzt, falls Operand = 0, sonst rückgesetzt.
- N - Gesetzt, falls Operand < 0, sonst rückgesetzt.

Adressierungsarten für Operand-1: (ea)

Dn	(An)+	d(An,Ri)	.
.	-(An)	Abs.W	.
(An)	d(An)	Abs.L	.



OPAL-68000 Cross-Assembler 1.02 (C) - 1984 Nülle / IDA-Software Seite 002

## OPAL-Test Nr. 3

```

                                PAGE
                                ORG    START
0003E8 4E71                    NOP
                                EVEN
0003EA 41                      DC.B    'A'
0003EB 4E71                    NOP      ; ADR ungerade
                                EVEN
0003EE 4E71                    NOP      ; ADR gerade

                                ; Listing abschalten: (XLIST)
                                LIST    ; Listing wieder einschalten

0003F0 446965736573            DC.B    'Dieses ist ein String 1',STOP,12,12X8,12H
0003F6 206973742065
0003FC 696E20537472
000402 696E67202100
000408 0C0A12
00040B 446965736573            DC.B    'Dieses ist ein String 1',STOP,12,12X8,12H
000411 206973742065
000417 696E20537472
00041D 696E67202100
000423 0C0A12
000426 446965736573            DC.B    'Dieses ist ein String 1',STOP,12,12X8,12H
00042C 206973742065
000432 696E20537472
000438 696E67202100
00043E 0C0A12

                                XFUNCH ; Maschinen-Code nicht mehr ausdrucken:
                                |-----
000441 4C.B    'Dieses ist ein String 1',STOP,12,12X8,12H
00045C 4C.B    'Dieses ist ein String 1',STOP,12,12X8,12H
000477 4C.B    'Dieses ist ein String 1',STOP,12,12X8,12H
000492 NOP
                                PUNCH  ; ab hier Maschinen-Code wieder drucken:
                                |-----

000494 4E71                    NOP

```

OPAL-68000 Cross-Assembler 1.02 (C) - 1984 Wilke / IDA-Software Seite 003

## OPAL-Test Nr. 3

```

                                PAGE
                                | File-Include
000496 4E71                    NOP                    | Main
000498 4E71                    NOP                    | Male
A                                INCLUDE '123456789'      | Fehler: Filename zu lang
A                                INCLUDE '12345678.1234'   | Fehler: Extent zu lang
A                                INCLUDE 'C:123456789'     | Fehler: Filename zu lang
A                                INCLUDE '0:12345678.1234' | Fehler: Extent zu lang
A                                INCLUDE '1:ABC'           | Fehler: Orgue unzuellaessig
A                                INCLUDE '1234:34'         | Fehler: unzuellaessiges Zeichen
A                                INCLUDE '123.123.12'      | Fehler
A                                INCLUDE ''                | Fehler: Filename zu kurz

                                C
00043A 4E71                    C                        | INCLUDE '11.m68'      | File 1
                                C                        NOP
00049C 4E71                    C                        NOP
00049E 4E71                    C                        NOP
0004A0 4E71                    C                        NOP | Ende File 1
                                C                        INCLUDE 'a:i2.m68' | File 2
U 0004A2 66000000              C R                     BNE LABEL
0004A6 4E71                    C                        NOP
                                C                        INCLUDE 'a:i3.m68' | File 3
                                C
                                C |
P                                C                        INCLUDE 'A:i2.M68' | *** Schachtelungs-Fehler
D                                INCLUDE 'e:i4.m68'         | File 4 *** nicht vorhanden
D                                INCLUDE 'o:i5.m68'         | File 5 *** nicht vorhanden
0004A8 4E71                    NOP                    | Male

```

OPAL-68000 Cross-Assembler 1.02 (C) - 1984 Wille / IDA-Software Seite 004

## OPAL-Test Nr. 3

```

                                PAGE
                                ; SIZE explicit and by Default:
                                DC.W  'Dieses ist ein String !',STOP,12,12x8,12H

0004A4 446965736573
0004B0 206973742065
0004B6 696E20537472
0004BC 696E67202100
0004C2 0000C000A00
0004C8 12
0004C9 446965736573      DC.L  'Dieses ist ein String !',STOP,12,12x8,12H
0004CF 206973742065
0004D5 696E20537472
0004DB 696E67202100
0004E1 0000000000
0004E7 0C000000A00
0004ED 000012

                                SIZE.L
                                DC  'Dieses ist ein String !',STOP,12,12x8,12H

0004F0 446965736573
0004F6 206973742065
0004FC 696E20537472
000502 696E67202100
000508 0000000000
00050E 0C000000A00
000514 000012

                                SIZE.W
                                DC  'Dieses ist ein String !',STOP,12,12x8,12H

000517 446965736573
00051D 206973742065
000523 696E20537472
000529 696E67202100
00052F 0000C000A00
000535 12

                                SIZE.B
                                DC  'Dieses ist ein String !',STOP,12,12x8,12H

000536 446965736573
00053C 206973742065
000542 696E20537472
000548 696E67202100
00054E 0C00A12
000551 4E71      NOP
000553 4E71      NOP

```

OPAL-68000 Cross-Assembler 1.02 (C) - 1984 Mitke / IDA-Software Seite 005

# OPAL-Test Nr. 3

```

                                PAGE
                                ; Flags werden ausgedruckt:
                                ;-----
000555 4E71          LAB_1: NOP
000557 60FC          BFA      LAB_1
000559 4E71      N  LAB_2: NOP
00055B 6400FFF0      R      BCC.W LAB_1
00055F 4E71      N  LAB_3: NOP

                                XFLAG
                                ; keine flags mehr ausdrucken:
                                ;-----
000561 4E71          LAB_4: NOP
000563 60FC          BFA      LAB_4 ; R-flag
000565 4E71          LAB_5: NOP      ; N-flag
000567 6400FFF0      BCC.W LAB_4      ; R-flag
00056B 4E71          LAB_6: NOP      ; N-flag
                                FLAG
                                ; flags wieder drucken
                                ;-----
00056D 4E71      N  LAB_7: NOP

                                TITLE 'neue titelseite, OPAL-Test-3; PSEUDOs'
```

OPAL-68000 Cross-Assembler 1.02 (C) - 1984 Milke / IDA-Software Seite 006

neue Titelseite, OPAL-Test 3: PSEUDO:

```

                                PAGE          ; neuer Seiten-Anfang
                                ;
                                ; Test Conditional-Assembly, Bedingung ist erfuellt:
                                ;=====
                                IFE          0
00056F 4E71                      NOP
                                ENDF

                                IFN          9876
000571 4E71                      NOP
                                ENDF

                                IFP          8352
000573 4E71                      NOP
                                ENDF

                                IFM          -5555
000575 4E71                      NOP
                                ENDF

                                ; Bedingung nicht erfuellt:
                                ;=====
                                IFE          01010111X2
                                NOP
                                ENDF

                                IFN          00000XB
                                NOP
                                ENDF

                                IFP          -1234567XB
                                NOP
                                ENDF

                                IFM          5555
                                NOP
                                ENDF

```

OPAL-68000 Cross-Assembler 1.02 (C) - 1984 Mike / IDA-Software Seite 007

new Titelzeile, OPAL-Test-3: PSEUDO's

		PAGE	
		verschachtelt:	- Schachtelungstiefe: -
000577	4E71	IFE	0
		NOP	1
		IFN	1
000579	4E71	NOP	2
		IFP	2
00057B	4E71	NOP	3
		IFN	-3
00057D	4E71	NOP	4
		IFE	0
00057F	4E71	NOP	5
		IFN	0
		NOP	Bedingung nicht erfüllt
		NOP	6
		IFP	3
		NOP	7
		IFN	-07
		NOP	0
		IFE	0
		NOP	3
		ENDIF	
		NOP	0
		ENDIF	
		NOP	7
		ENDIF	
		NOP	6
		ENDIF	
000581	4E71	NOP	5
		ENDIF	
000583	4E71	NOP	4
		ENDIF	
000585	4E71	NOP	3
		ENDIF	
000587	4E71	NOP	2
		ENDIF	
000589	4E71	NOP	1
		ENDIF	
00058B	4E71	NOP	
P		ENDIF	0000 Fehler: ENDIF ohne IF
00058D	4E71	NOP	
P		ENDIF	0000 Fehler: ENDIF ohne IF



DPAL-68000 Cross-Assembler 1.02 (C) - 1984 Wilke / IDA-Software Seite 008

neue Titelzeile, DPAL-Test-3: PSEUDOs

```

PAGE
ORG 'ABC' ; ... leicht zu finden in '.COO'-File
FILL '' ; Füll-Zeichen fuer 'DS'-Anweisung
SIZE.L ; Default SIZE is LONG
414243 DS 10H ; makes 16 LONG-Spaces (filled with '')
FILL 'a'
SIZE.W
414287 DS 10H ; makes 16 WORD-Spaces (filled with 'a')
FILL '-'
SIZE.B
414263 DS 10H ; makes 16 BYTE-Spaces (fills with '-')

; explizite SIZE-Angaben:
FILL 'A' ; füll-Zeichen fuer 'DS'-Anweisung
4142B3 DS.L 10H ; makes 16 LONG-Spaces (filled with 'A')
FILL 'B'
4142F3 DS.W 10H ; makes 16 WORD-Spaces (filled with 'B')
FILL 'C'
414313 DS.B 10H ; makes 16 BYTE-Spaces (fills with 'C')
414323 4E71 NOP

```

OPAL-68000 Cross-Assembler 1.02 (C) - 1984 Wille / IDA-Software Seite 009

new Titelzeile, OPAL-Test-3: PSEUDOS

```

                                PAGE
                                1
                                verschiedene EQU's:
43444546 M ABCDEF: EQU 'ABCDEF'
42434445 M ABCDE: EQU 'ABCDE'
41424344 M ABCD: EQU 'ABCD'
00414243 M ABC: EQU 'ABC'
00004142 M AB: EQU 'AB'
00000041 M A: EQU 'A'
00000000 M X: EQU ''

010A6001 SYMBOL_1:EQU 01110110100110101111010001X2 ; binär
FE25942F M SYMBOL_2:EQU ~01110110100110101111010001X2 ; binär
1A3F5001 M SYMBOL_3:EQU 76543217654321X8 ; octal
07B60505 M SYMBOL_4:EQU ~'ABCDEF GHIJK' ; ASCII
45960202 M SYMBOL_5:EQU 1234567890 ; decimal
0545440C M SYMBOL_6:EQU 5454ABC ; hexa
12345670 SYMBOL_7:EQU 12345670H ; hexa
12345670 M SYMBOL_8:EQU SYMBOL_9
12345670 SYMBOL_9:EQU SYMBOL_7
00414325 M SYMBOL_10:EQU $ ; PC

M 00000001 SEHR_LANGE3_SYMBOL_1: EQU 1 ; 0000 multiple-def
M 00000002 SEHR_LANGE3_SYMBOL_2: EQU 2 ; 0000 multiple-def
M 00000003 SEHR_LANGE3_SYMBOL_3: EQU 3 ; 0000 multiple-def
M 00000004 SEHR_LANGE3_SYMBOL_4: EQU 4 ; 0000 multiple-def

M 414325 D63C0001 ADD @SEHR_LANGE3_SYMBOL_4,03 ; 0000 multiple-def
M 414329 D63C0001 ADD @SEHR_LANGE3_SYMBOL_3,03 ; 0000 multiple-def

00000001 - SYMBOL_1:REDEF 1
414320 D63C0001 ADD @SYMBOL_1,03
00000002 - SYMBOL_1:REDEF 2
414331 D63C0002 ADD @SYMBOL_1,03
00000003 - SYMBOL_1:REDEF 3
414335 D63C0003 ADD @SYMBOL_1,03

```

OPAL-68000 Cross-Assembler 1.02 (C) - 1984 Wilke / IDA-Software Seite 818

neue Titelzeile, OPAL-Test-3: PSEUDOs

A	00000241	M equ	AB	00000142	N equ
ABC	00414243	N equ	ABCD	41424344	N equ
ABCDE	42434445	M equ	ABCDEF	43444546	N equ
BELL	00000007	equ	B10_MINUS	FF054320	N equ
B10_PLUS	12345678	M equ	CR	00000000	equ
DUMMY	0000040C	N input	EA	00000058	equ
ESC	00000010	equ	LAB_1	00000555	label
LAB_2	00000559	N label	LAB_3	0000055F	M label
LAB_4	00000561	label	LAB_5	00000563	M label
LAB_6	00000568	M label	LAB_7	00000560	N label
LF	0000000A	equ	MEDIUM_MINUS	FFFF000E	N equ
MEDIUM_PLUS	00003456	N equ	SEMI_LARGES_	00000001	M equ
SEITEN_ABSTA	0000000E	equ	SEITEN_INHAL	0000003A	equ
SMALL_MINUS	FFFFFFFAB	N equ	SMALL_PLUS	00000000	N equ
START	000003E0	input	STOP	00000000	equ
SYMBOL_1	00000003	redef	SYMBOL_10	00414323	M equ
SYMBOL_2	FE25342F	N equ	SYMBOL_3	1A3F5001	M equ
SYMBOL_4	07B68505	N equ	SYMBOL_5	49360202	N equ
SYMBOL_6	0545400C	N equ	SYMBOL_7	12345678	equ
SYMBOL_8	12345678	N equ	SYMBOL_9	12345678	equ
X	00000000	N equ	ZEILEN_LADN	0000003F	equ

0021 fehlerhafte Zeile(n)

OPAL-68000 Cross-Assembler 1.02 (C) - 1984 Mitke / IDA-Software Seite 001

Hallo

```
TITLE 14,'Halle'
LIMIT 15
LINE 131
```

```
;
; Name : Hello.M68
; Typ : OPAL 68000 Source Code
; Stand : 01.09.84
;
; Hinweis : Benutzung von CPM Systemfunktionen
; Initialisierung fuer Drecker EPSON FX 80
```

;Programm zur Ausgabe eines Strings

```
;Vereinbarungen
00000000 CR: EQU DDH ;Carriage Return
0000000A LF: EQU DAH ;Line Feed
00000024 EOT: EQU '0' ;Kennung Textende

00000000 BDOS: EQU 0 ;CP/M Funktionsaufruf
00000009 PRINTSTRING: EQU 9 ;CP/M Funktion 9

ORG 1000H ;Programm beginnt ab Adresse 1000H

001000 41FA00A M START: LEA.L TEXT(0),A0 ;Startadresse des Strings
; nach Register A0 laden
; Adressierungsart PC relativ
001004 7609 MOVEQ #PRINTSTRING,03 ;Funktionsnummer nach Register D3 laden
001006 0E40 TRAP #BDOS ;Systemaufruf --> Funktion ausfuehren

001000 4203 CLR.B D3 ;LSB in Register D3 loeschen
; --> Funktion 0 = zurueck in's System
00100A 0E40 TRAP #BDOS ;funktion ausfuehren

;Durch einen Systemaufruf ueber TRAP 00 mit der Funktionsnummer 0 wird das
;User-Program verlassen und das Betriebssystem angesprochen. Man kehrt also
;entweder in den Debugger (Aufruf des Programmes im Debugger), oder aber in's
;CP/M (Aufruf des Programmes vom Run-Time-Simulator) zurueck.

00100C 000A TEXT: DC.B CR,LF ;Carser auf neue Zeile setzen
00100E 477574636E20 DC.B 'Beten Tag' ;dieser Text wird auf der Console ausgegeben
001014 546167 DC.B EOT ;Ende Kennung des Strings
001017 24
```

Keine Assembler-Fehler

---

Benutzer - Kommentar

---

Wir sind an Ihrer Meinung über dieses Produkt interessiert!  
Wenn Sie also Anregungen, Kritik oder Verbesserungsvorschläge  
zu den Programmen oder der Dokumentation haben schreiben Sie uns.

Mir liegt folgende Programm-Version vor: \_\_\_\_\_

---

Bemerkungen zu Programm/Dokumentation:

OPAL-68000

---

Bemerkungen zu Programm/Dokumentation:

RSU-68000

---

Bemerkungen zu Programm/Dokumentation:

HDT-68000

---

• bitte senden an: Inq.-Büro Wilke, Postfach 1727, D-5100 Aachen 1

---



**Das 68000-Paket erhalten Sie bei**